ACTIAN™

Distributed Tuning Interface Guide

Zen v15

Activate Your Data™

Contents

About This Document

This documentation covers the Zen Distributed Tuning Interface components provided in its Software Development Kit (SDK).

Who Should Read This Manual

This document is designed for any user who is familiar with Zen and wants to develop administrative applications using the Distributed Tuning Interface.

This manual does not provide comprehensive usage instructions for the software or instructions for using other database access methods. It does provide a reference for using the Distributed Tuning Interface.

We would appreciate your comments and suggestions about this document. Your feedback can determine what we write about the use of our products and how we deliver information to you. Please post your feedback in the community forum on the Zen website.

Distributed Tuning Interface Guide

The following topics introduce the Zen Distributed Tuning Interface:

- Overview of Distributed Tuning Interface
- Basics Of Using DTI
- Sample Programs For DTI
- Common Tasks With DTI

You can also go directly to Distributed Tuning Interface Reference for detailed information on the use of DTI in Zen.

Overview of Distributed Tuning Interface

The purpose of Distributed Tuning Interface (DTI) is to provide an application programming interface for configuration, monitoring, and diagnostics of Zen components.

Note: For brevity, throughout the rest of this manual Distributed Tuning Interface is referred to by as DTI.

String Arguments Encoding

A user application uses the client's OS encoding at the API level. DTI handles internally the differences between OS encodings on the server and client.

If an older client is communicating with the server, the database engine assumes that the client is using an encoding compatible with those available on the server.

API Categories

The categories of available APIs are summarized in DTI Function Groups.

Execution Privileges

Generally, you want your DTI application to be able to call any of the DTI functions and view or modify all configuration settings. To ensure this full access, connect to the server by providing a name and password of a user with administrative level privileges on the server machine. This applies if the DTI application is running locally through a Terminal Services session or running

remotely. An application running locally can omit the user name and password and still be able call any of the DTI functions and view or modify all configuration settings. See Making a Connection to a Server Using DTI.

Without administrator level privileges, an application running locally through a Terminal Services session or running remotely returns an access error for most of the DTI functions. Only a subset of the functions work. For example, many of the functions that can modify configuration settings when full access is permitted are restricted to read-only access.

Basics Of Using DTI

Header Files

The DTI functions are defined in the following header files:

- btitypes.h
- catalog.h
- config.h
- connect.h
- ddf.h
- dticonst.h
- dtilicense.h
- dtisecurity.h
- monitor.h

Link Libraries

The following table lists the link libraries for DTI and the release version in which the library was first available on Windows, Linux, and macOS. Link your application to the appropriate library as defined in the table.

Library[1]	Windows	Linux	macOS	Version of Library First Available
w3dbav90.lib[2]	32-bit			PSQL v9.0
w64dba.lib	64-bit			PSQL v10.0

Library[1]	Windows	Linux	macOS	Version of Library First Available
w3dbav80.lib[2]	32-bit			Pervasive.SQL V8.0
w3dbav78.lib[2]	32-bit			Pervasive.SQL 2000i (SP3)
w3dbav75.lib[2]	32-bit			Pervasive.SQL 2000
libpsqldti.so		32-bit		Pervasive.SQL V8.5
libpsqldti.so		64-bit		PSQL 10.10
libpsqldti.dylib			64-bit	PSQL v12.01

[1] All libraries have been compiled with Microsoft Visual Studio 2015.

[2] Each 32-bit library is a superset of the previous library. For example, w3dbav90.lib is a superset of w3dbav75.lib, w3dbav78.lib, and w3dbav80.lib.

The functions for the DTI are documented in Distributed Tuning Interface Reference.

Before Calling Any Functions

When you want to invoke DTI, you must first call the PvStart() function. Then you can call multiple DTI functions before ending the session.

When ending a session, you must call PvStop() to close the session.

Sample Programs For DTI

By default, the runtime files for the DTI access method are installed with the Zen database engine and with Zen Client. At a minimum, you need Zen Client to create a DTI application.

The header files and sample files are available by online download. Sample files pertaining to a particular development environment are installed in separate directories, as shown in the following table.

Development Environment	Location
MS Visual C++ 8	*install_location*\SAMPLES\MSVC2005
MS Visual C++ 7	*install_location*\SAMPLES\MSVC2003
MS Visual C++ 6	*install_location*\SAMPLES\MSVC
Delphi 5	*install_location*\SAMPLES\DELPHI5

For additional information, see the DTI release notes (readme_dti.htm) installed with the Zen database engine.

Common Tasks With DTI

This topic outlines key tasks that are often used with DTI.

Making a Connection to a Server Using DTI

This documents the procedure for obtaining a connection handle to a server, which is a first step for many DTI functions.

To obtain a Connection handle to a server

1. Start a DTI session

```
// initialize status code return
BTI_LONG status = 0;
// Call PvStart function with its reserved
// parameter
status = PvStart(0);
```

2. Connect to a server

```
// initialize variables
BTI_LONG status = 0;
BTI_CHAR_PTR uName = "jsmith";
BTI_CHAR_PTR pword = "123";
BTI_CHAR_PTR svrName = "myserver";
BTI_LONG hConn = 0xFFFFFFFF;
// after execution, hConn contains connection
// handle to pass to other functions
status = PvConnectServer(svrName, uName, pword, &hConn);
// if status != 0, handle errors now
```

Connection handles are required by many DTI functions. You can have multiple connections open at a time. For each connection or handle, however, you should call the PvDisconnect() function to release the handle.

```
status = PvDisconnect(phConn);
```

Obtaining a Setting ID Using DTI

Many of the configuration functions take a setting ID as a parameter. This procedure describes the prerequisite functions for obtaining a setting ID.

To obtain the ID for a Specific Setting

1. Perform the procedure Making a Connection to a Server Using DTI to obtain a connection handle.
2. Using the connection handle returned by PvConnectServer(), obtain a list of categories by calling PvGetCategoryList().
3. For each category, get the list of settings using PvGetSettingList() and the settings count using PvGetSettingListCount().
4. Scan for the setting that you want.
5. Retrieve information about the setting using PvGetSettingInfo().
6. When done, disconnect from the server by calling PvDisconnect().
7. End the DTI session by calling PvStop().

Passing a DTI Structure as a Parameter

Many functions require that you pass a DTI structure when making the functional call. The following code segment shows an example of a function call including a structure. See DTI Structures for more information about DTI structures.

```
WORD rValue = P_OK;
TABLEMAP* tableList;
WORD tableCount;
rValue = PvGetTableNames(m_DictHandle, &tableList, &tableCount);
```

Distributed Tuning Interface Reference

The purpose of DTI is to provide an interface for configuring, monitoring, and diagnosing Zen components. DTI provides the functionality of Zen utilities from within your application.

The following topics cover the interface and its use:

Using the DTI Function Reference

For each function, the following information is provided:

- Brief description – provides a short description of the function.
- Syntax – shows the C prototype syntax for the function.
- Arguments – provides detailed descriptions of the function arguments, and indicates which values are modified by the function. Parameters marked "in" are input-only, not modified by the function. Parameters marked "out" contain values modified by the function. Parameters marked "in/out" contain values that are both used by the function as input and modified by the function.
- Return Values – lists the possible return values and their meanings.
- Remarks – provides additional explanation about a function's parameters, effects, or usage.
- Example – provides a sample code segment showing the function's use.
- See Also – lists related functions and topics.

DTI Function Groups

The Distributed Tuning Interface is divided into function groupings. For a summary of these groupings, please see the following table. The function descriptions begin in the following section in alphabetical order.

Function Group	Purpose	List of Functions
Catalog catalog.h	Managing the database catalog information, such as creating, opening, copying, or closing named databases, and creating, modifying or deleting data source names (DSNs),	PvCheckDbInfo() PvCloseDatabase() PvCopyDatabase() PvCountDSNs() PvCreateDatabase() PvCreateDatabase2() PvCreateDSN() (deprecated) PvCreateDSN2() (deprecated) PvDeleteDSN() (deprecated) PvDropDatabase() PvFreeDbNamesData() PvGetDbCodePage() PvGetDbDataPath() PvGetDbDictionaryPath() PvGetDbFlags()) PvGetDbName() PvGetDbNamesData() PvGetDbServerName() PvGetDSN() (deprecated) PvGetDSNEx() (deprecated) PvGetDSNEx2() (deprecated) PvGetEngineInformation() PvListDSNs() (deprecated) PvModifyDatabase() PvModifyDatabase2() PvModifyDSN() (deprecated) PvModifyDSN2() (deprecated) PvOpenDatabase()

Function Group	Purpose	List of Functions
Configuration config.h	Controlling the configuration settings for the database engines, the communication managers, and the local requester components.	PvCountSelectionItems() PvGetAllPossibleSelections() PvGetBooleanStrings() PvGetBooleanValue() PvGetCategoryInfo() PvGetCategoryList() PvGetCategoryListCount() PvGetLongValue() PvGetSelectionString() PvGetSelectionStringSize() PvGetSelectionValue() PvGetSettingHelp()() PvGetSettingHelpSize() PvGetSettingInfo() PvGetSettingList() PvGetSettingListCount() PvGetSettingMap() PvGetSettingUnits() PvGetSettingUnitsSize() PvGetStringType() PvGetStringValue() PvGetStringValueSize() PvGetValueLimit() PvIsSettingAvailable() PvSetBooleanValue() PvSetLongValue() PvSetSelectionValue() PvSetStringValue()
Connection connect.h	Starting and stopping a DTI session, connecting to a server, retrieving the name of the connected server, and disconnecting from a server.	PvConnectServer() PvDisconnect() PvGetServerName() PvStart() PvStop()

Function Group	Purpose	List of Functions
Dictionary ddf.h	Creating and closing dictionaries (DDFs), and creating or deleting tables, indexes, users and groups.	PvAddIndex() PvAddTable() PvAddUserToGroup() PvAlterUserName() PvAlterUserPassword() PvCloseDictionary() PvCreateDictionary() (deprecated) PvCreateGroup() PvCreateUser() PvDropGroup() PvDropIndex() PvDropIndexByName() PvDropTable() PvDropUser() PvFreeTable() PvFreeTableNames() PvGetError() PvGetTable() PvGetTableNames() PvGetTableStat() PvGetTableStat2() PvGetTableStat3() PvOpenDictionary() (deprecated) PvRemoveUserFromGroup()
License Administration dtilicense.h	Administering licensing such as authorizing or deauthorizing a key or retrieving information about keys.	PvAddLicense() PvValidateLicenses() PvDeleteLicense() PvGetProductsInfo()

Function Group	Purpose	List of Functions
Monitoring and Diagnostic monitor.h	Monitoring files, clients, and SQL connections, such as the following information for the MicroKernel Engine: Active Files – count and list open files, query if file is open, query user who opened/locked the file, obtain page size, read-only flag, record locks, transaction locks, number of handles, obtain handle information for each handle. Active Clients – count and list clients, query active handles, obtain client information, obtain handle information, disconnect a client and all client functionality. Resource Usage – retrieve current, peak, and maximum settings for data, including files, handles, clients, worker threads, licenses in use, transactions, locks. Communications Statistics – retrieve all communications statistics, total, delta, current, peak, maximum where appropriate, reset delta functionality.	PvDisconnectMkdeClient() PvDisconnectSQLConnection() PvFreeMkdeClientsData() PvFreeOpenFilesData() PvFreeSQLConnectionsData() PvGetFileHandlesData() PvGetFileHandleInfo() PvGetFileInfo() PvGetMkdeClientId() PvGetMkdeClientInfo() PvGetMkdeClientHandlesData() PvGetMkdeClientHandleInfo() PvGetMkdeClientsData() PvGetMkdeCommStat() PvGetMkdeCommStatEx() PvGetMkdeUsage() PvGetMkdeUsageEx() PvGetMkdeVersion() PvGetOpenFilesData() PvGetOpenFileName() PvGetSQLConnectionsData() PvGetSQLConnectionInfo()
Security dtisecurity.h	Enabling, disabling, or querying the status of security on databases.	PvIsDatabaseSecured() PvSecureDatabase() PvUnSecureDatabase()

DTI Error Messages

Refer to dticonst.h and ddfstrct.h for the defined status codes.

DTI Structures

The following describes the structures used in DTI. Each structure grouping details the type of structures included and any notable settings or arguments that may be required. Structures are stored in the following header files:

- CONFIG.H
- DDFSTRCT.H
- MONITOR.H

For detailed information specific to each structure, refer to the corresponding header file for that structure.

CONFIG.H Structures

The following lists the structures included in CONFIG.H. For detailed information about any of these structures, refer to the config header file.

- PVCATEGORYINFO
- PVSETTINGINFO

DDFSTRCT.H Structures

The following lists the structures included in DDFSTRCT.H. For detailed information about any of these structures, refer to the ddf header file.

- TABLEMAP
- TABLEINFO
- TABLEINFO Flags

```
B_FLAG_TRUE_NULLABLE = 64
```

  Table is true nullable. When the table is created, a one byte null indicator is added before each column that is nullable.

- TABLESTAT

  The systemDataKey (later systemData) field has a value of 0 if no system data is present and a value of 1 if system data or system data v2 is present.

- TABLESTAT2

  See Differences Between TABLESTAT2 and TABLESTAT.

- TABLESTAT3

  See Differences Between TABLESTAT3 and TABLESTAT2.

- COLUMNMAP
- COLUMNMAP Flags

```
B_FLAG_CASE_SENSITIVE = 1
```

Column values are case sensitive on comparisons and as part of index segments.

```
B_FLAG_NULLABLE = 4
```

If the table is created as true nullable, then a one byte null indicator column is added before the column value to indicate whether the column value is null.

```
B_FLAG_NTEXT = 2048
```

If a column is created as B_TYPE_BLOB, the data is treated as wide-character rather than character data.

```
B_FLAG_BINARY = 4096
```

If a column is created as B_TYPE_STRING or B_TYPE_BLOB, the data is treated as binary rather than character data.

- COLUMNMAP Data Types

  COLUMNMAP DataType can take the following values:

```
B_TYPE_STRING = 0,
B_TYPE_INTEGER = 1,
B_TYPE_FLOAT = 2,
B_TYPE_DATE = 3,
B_TYPE_TIME = 4,
B_TYPE_DECIMAL = 5,
B_TYPE_MONEY = 6,
B_TYPE_LOGICAL = 7,
B_TYPE_NUMERIC = 8,
B_TYPE_BFLOAT = 9,
B_TYPE_LSTRING = 10,
B_TYPE_ZSTRING = 11,
B_TYPE_NOTE = 12,
B_TYPE_LVAR = 13,
B_TYPE_BINARY = 14,
B_TYPE_AUTOINC = 15,
B_TYPE_BIT = 16,
B_TYPE_NUMERSTS = 17,
B_TYPE_NUMERSA = 18,
B_TYPE_CURRENCY = 19,
B_TYPE_TIMESTAMP = 20,
B_TYPE_BLOB = 21,
B_TYPE_GDECIMAL = 22,
B_TYPE_WSTRING = 25,
B_TYPE_WZSTRING = 26,
B_TYPE_GUID = 27,
B_TYPE_DATETIME = 30
```

- INDEXMAP
- INDEXMAP Flags

```
B_FLAG_DUPLICATES = 1
```

Duplicates allowed in index.

```
B_FLAG_MODIFIABLE = 2
```

Index is modifiable.

```
B_FLAG_SORT_DESCENDING = 64
```

Sort index descending.

```
B_FLAG_PARTIAL = 512
```

Index is partial. Partial Index flags on segments that are not the last segment in the index, are ignored. Partial Indexes only apply to the last segment in an index.

Differences Between TABLESTAT2 and TABLESTAT

Note the following differences between the TABLESTAT2 structure and the TABLESTAT structure:

- The fields for **tableName** and **tableLocation** allow more characters.
- The **numberOfRecords** field increases from 16 bits to 32 bits.
- File attribute fields were previously characters with values of "Y" or "N" to indicate whether the attribute is present or not. Attribute fields are now single byte integers with values of 1 or 0. A value of 1 means the attribute is present.
- The **freespaceThreshold** field is now an integer data type.
- The field **fileVersion** is no longer a float data type. It is now a single-byte integer that contains the same value that the Btrieve Stat (15) operation would return. For the 9.5 file format, the value returned is 0x95.
- A new field, **pageCompression**, indicates whether the physical file associated with the table has compressed pages or not.
- Previous fields **dataCompression** and **systemDataKey** have been renamed to **recordCompression** and **systemData**, respectively.

Differences Between TABLESTAT3 and TABLESTAT2

Note the following differences between the TABLESTAT3 structure and the TABLESTAT2 structure:

- The **numberOfRecords** field increases from 32 bits to 64 bits.

Backwards Compatibility

Zen clients can still make PvGetTableStat calls to the database engine. The database engine converts the reply message to a TABLESTAT2 structure or to a TABLESTAT structure as required based on the version of the client.

A Zen client determines the version of the database engine to which the client is connected. If the database engine version is prior to the current release, then PvGetTableStat2 returns a TABLESTAT structure and sets the value returned for pageCompression to 0.

MONITOR.H Structures

The following lists the structures included in MONITOR.H. For detailed information about any of these structures, refer to the monitor header file.

- PVDATETIME
- PVFILEINFO
- PVFILEHDLINFO
- PVCLIENTID
- PVMKDECLIENTINFO
- PVMKDECLIENTHDLINFO
- PVMKDEUSAGE
- PVMKDEUSAGEEX
- PVVERSION
- PVCOMMSTAT
- PVCOMMSTATEX
- PVCOMMPROTOCOLSTAT
- PVSQLCONNINFO
- PVSQLCONNID

DTI Calling Sequence

All Distributed Tuning Interface calls must initialize a DTI session by first calling PvStart().

```
status = PvStart(0);
// insert multiple DTI function calls here
status = PvStop(0);
```

The Remarks section of every function lists additional prerequisites and post requisites for that particular function.

DTI Function Definitions

This topic provides an alphabetical reference to the DTI functions.

PvAddIndex()

Adds indexes specified in *indexList* to the existing table and to the underlying data file.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvAddIndex(
   WORD             dictHandle,
   LPCSTR           tableName,
   INDEXMAP*        indexList,
   WORD             indexCount);
```

Arguments

In	*dictHandle*	Handle of an open dictionary returned by PvOpenDatabase().
In	*tableName*	Name of the table where the indexes will be added.
In	*indexList*	Array of index definitions.
In	*indexCount*	Number of indexes in the *indexList* array.

Return Values

PCM_Success	The operation completed successfully.
PCM_errFailed	The operation did not complete successfully.
PCM_errInvalidDictionaryHandle	The specified dictionary handle does not exist.
PCM_errTableNotFound	The specified table was not found.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errInvalidIndexName	The specified index name is invalid.
PCM_errColumnNotFound	The specified column was not found in the table.

Remarks

You must first open a dictionary successfully using PvOpenDatabase().

The table specified by *tableName* must exist in the dictionary specified by *dictHandle*.

You will need to allocate and release INDEXMAP array used to describe the indexes.

See Also

PvStart()
PvOpenDatabase()
PvDropIndex()
PvDropIndexByName()
PvCloseDictionary()
PvStop()

PvAddLicense()

Applies (authorizes) the specified license from the computer indicated by the connection.

Header File: dtilicense.h (See also Header Files)

Function First Available In Library: w3dbav80.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvAddLicense(
   BTI_LONG         hConnection,
   BTI_CHAR_PTR     license);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*license*	License to be applied (authorized).

Return Values

P_OK	The operation completed successfully.
P_E_FAIL	The operation did not complete successfully.
P_E_LIC_ALREADY_INSTALLED	The license is already applied.
P_E_LIC_INVALID	The license specified is invalid.
Status code pertaining to license administration or to authorization	See *Status Codes and Messages* for License Administrator Status Codes and Authorization Status Codes.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Example

```
BTI_CHAR_PTR add_lic = "ERXVD3U4ZS9KR94QPDHV5BN2";
status = PvAddLicense(P_LOCAL_DB_CONNECTION, add_lic);
```

See Also

PvValidateLicenses()
PvDeleteLicense()
PvGetProductsInfo()
PvStop()
PvStart()

PvAddTable()

Creates a new table in the existing dictionary and a data file at the location specified in the table properties.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvAddTable(
   WORD             dictHandle,
   TABLEINFO*       tableProps,
   COLUMNMAP*       columnList,
   WORD             columnCount,
   INDEXMAP*        indexList,
 WORD             indexCount);
```

Arguments

In	*dictHandle*	Handle of an open dictionary returned by PvOpenDatabase().
In	*tableProps*	Structure containing table information.
In	*columnList*	Array of columns defined in the table.
In	*columnCount*	Number of columns in columnList.
In	*indexList*	Array of index definitions.
In	*indexCount*	Number of indexes in the following indexList array.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidDictionaryHandle	The specified dictionary handle does not exist.
PCM_errTableNotFound	The specified table was not found.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errInvalidColumnName	The specified column name is invalid.

PCM_errInvalidDataType	The specified data type is invalid.
PCM_errDuplicateColumnName	The column name already exists in the table.
PCM_errInvalidDataSize	The data size is invalid.
PCM_errInvalidIndexName	Index name is invalid.
PCM_errColumnNotFound	Column specified for a segment cannot be found.

Remarks

You must first open a dictionary successfully using PvOpenDatabase().

This function has to be provided with table information, columns, and indexes. The *indexCount* and *indexList* parameters are optional because indexes are not required to create a table.

This function will fail if a table with the same name is already present in the specified dictionary.

Table properties must be set up correctly and an array of at least one column must be passed.

You will need to allocate and release COLUMNMAP and INDEXMAP arrays and TABLEINFO structure used to describe table. See also COLUMNMAP Flags.

The offset of a field within its row can be accessed through the PvGetTable() function. The COLUMNMAP structure has been modified in ddfstrct.h to contain this additional information. This new field is ignored when calling the PvAddTable() and PvFreeTable() functions. Refer to ddfstrct.h and ddf.h.

See Also

PvStart()
PvOpenDatabase()
PvGetTableNames()
PvFreeTableNames()
PvDropTable()
PvCloseDictionary()
PvStop()

PvAddUserToGroup()

Adds an existing user to an existing group in the database.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvAddUserToGroup(
   BTI_WORD            dbHandle,
   const BTI_CHAR*     user,
   const BTI_CHAR*     group);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*user*	Database user name
In	*group*	Database group name

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The specified account or user name does not exist.
PCM_errUserAlreadyPartOfGroup	User already part of the group.
PCM_errDatabaseHasNoSecurity	Database has no security.
PCM_errSessionSecurityError	Database opened with insufficient privilege.

Remarks

This function will fail if the specified group or user do not already exist in the database, or if the user is a member of another group.

The following preconditions must be met:

- You must first open a database successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.
- The user and group already exist in the specified database.
- The user is not a member of another group.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvAlterUserName()
PvCreateGroup()
PvCreateUser()
PvDropGroup()
PvDropUser()
PvRemoveUserFromGroup()
PvOpenDatabase()
PvCloseDatabase()

PvAlterUserName()

Alters an existing user name in the specified database.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvAlterUserName(
   BTI_WORD            dbHandle,
   const BTI_CHAR*     user,
   const BTI_CHAR*     newName);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*user*	Database user name.
In	*newName*	New name for the database user. If set to NULL, the function fails.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The account or user name does not exist, or the new name is invalid.
PCM_errUserAlreadyExists	New user name already exists.
PCM_errDatabaseHasNoSecurity	Database has no security.
PCM_errSessionSecurityError	Database opened with insufficient privilege.

Remarks

This function will fail if *newName* is set to NULL, or if *newName* is already present in the database.

The following preconditions must be met:

- You must first open a dictionary successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.
- The user name must already exist in the specified database.
- The new user name cannot already exist in the specified database.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvAlterUserPassword()
PvAddUserToGroup()
PvRemoveUserFromGroup()
PvCreateUser()
PvDropUser()
PvOpenDatabase()
PvCloseDatabase()

PvAlterUserPassword()

Alters an existing user password.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvAlterUserPassword(
   BTI_WORD            dbHandle,
   const BTI_CHAR*     user,
   const BTI_CHAR*     newPassword);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*user*	Database user name.
In	*newPassword*	New user password. If set to NULL, the password is cleared.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The specified account or user name does not exist.
PCM_errDatabaseHasNoSecurity	Database has no security.
PCM_errSessionSecurityError	Database opened with insufficient privilege.

Remarks

The following preconditions must be met:

- You must first open a database successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.
- The user name must already exist in the specified database.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvAlterUserName()
PvAddUserToGroup()
PvRemoveUserFromGroup()
PvCreateUser()
PvDropUser()
PvOpenDatabase()
PvCloseDatabase()

PvCheckDbInfo()

Checks the consistency of a database.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvCheckDbInfo(
   BTI_LONG       hConnection,
   BTI_CHAR_PTR   dbName,
   BTI_ULONG      checkFlags);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of an existing named database. A list of all named databases for a particular server is obtained with the PvGetDbNamesData() function. A single named database from the resulting list can be obtained with the PvGetDbName() function.
In	*checkFlags*	Reserved. The function checks for all database flags.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Connection handle that identifies the server is invalid.
P_E_NULL_PTR	The function was called with a null pointer.
P_E_ACCESS_RIGHT	Insufficient access rights to call the function.
P_E_NOT_EXIST	Named database specified in *dbName* does not exist.
P_E_FAIL	A general failure occurred.

Remarks

If the database is consistent, then the return value for this function is P_OK. If the database is not consistent or if the function call fails, then the return value is one of the error codes listed above.

Example

```
BTI_WORD res; // returned value from function call
BTI_CHAR_PTR dbName;    // database name
BTI_ULONG        checkFlags;     // database flags
BTI_LONG        hConnection;    // connection handle
BTI_LONG         reserved;
  // reserved value for PvStart() and PvStop()

  // Initialize variables.
dbName   = "demodata";
  // The name of the database is demodata
checkFlags = 0xFFFFFFFF;   // Checks all flags
hConnection = P_LOCAL_DB_CONNECTION;
  // Set the connection handle to local connection

  // P_LOCAL_DB_CONNECTION is defined in config.h
reserved = 0;

 // Start a DTI session before making any DTI calls.
res = PvStart (reserved);

if (res == P_OK)
 {
    // DTI session started successfully.
    // You can now make multiple DTI calls here.

         res = PvCheckDbInfo (hConnection,
                              dbName,
                              checkFlags);

         if (res == P_OK)
           {
       // Database is consistent.
}
           else
            {
       // Put your code here to handle the error code
       // returned from PvCheckDbInfo ().
}
      // Close DTI session.
            Res = PvStop (&reserved);
}
```

See Also

PvStart()
PvConnectServer()
PvGetDbNamesData()
PvGetDbName()

PvFreeDbNamesData()
PvDisconnect()
PvStop()

PvCloseDatabase()

Closes an open database handle.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvCloseDatabase(
   BTI_WORD       dbHandle);
```

Arguments

In	*dbHandle*	Handle to a database opened by PvOpenDatabase().

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	Operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation
PCM_errDictionaryNotOpen	No database open with specified handle.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Valid database handle returned by PvOpenDatabase().

See Also

PvStart()
PvConnectServer()

PvOpenDatabase()
PvDisconnect()
PvStop()

PvCloseDictionary()

Closes an open dictionary.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvCloseDictionary(
   WORD              dictHandle);
```

Arguments

In	*dictHandle*	Handle of an open or newly-created dictionary.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errDictionaryNotOpen	The specified dictionary was not open.

Remarks

This function requires a handle for an open dictionary file, which can be obtained with the PvCreateDictionary() function.

Since multiple dictionaries can be open at one time, you need to call this function for every open or newly-created dictionary.

Example

```
PRESULT status = 0;
status = PvCloseDictionary(myDictionaryHandle);
```

See Also

PvStart()
PvOpenDatabase()
PvCreateDictionary()
PvStop()

PvConnectServer()

Attempts to connect to the target server that has the Zen database engine installed. If connection is established successfully, a connection handle is returned for subsequent references.

Header File: connect.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvConnectServer(
   BTI_CHAR_PTR        serverName,
   BTI_CHAR_PTR        userName,
   BTI_CHAR_PTR        password,
   BTI_LONG_PTR        phConnection);
```

Arguments

In	*serverName*	Server name or IP address to which you want to connect. See also Drive-based Formats in *Getting Started with Zen*.
In	*userName*	User name with which you will connect to *serverName*. See the Remarks section for information on omitting this parameter.
In	*password*	User password. See the Remarks section for information on omitting this parameter.
In/Out	*phConnection*	Address of a long integer that receives the connection handle if connection is successful.

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed to connect to the named server.
P_E_SERVER_NOT_FOUND	The specified server was not found
P_E_ENGINE_NOT_LOADED	The specified engine is not running.
P_E_REQUESTER_NOT_LOADED	The client requester is not loaded.

P_E_SERVER_TABLE_FULL	The internal server name table is full.
P_E_CLIENT_CONNECTIONS_LIMIT_REACHED	The operation could not connect because the limit on client connections has been reached. Check the configuration of the server.
P_E_PERMISSION_ERROR	The operation encountered a permissions error.
P_E_NO_MEMORY	The operation encountered a memory error.
P_E_NO_AVAILABLE_TRANSPORT	No remote connection could be established.
P_E_CONNECTION_LOST	The remote connection to the server was lost.

Remarks

You must know the name of the server to which you want to connect. You can have open connections to multiple servers.

An application running locally where the database engine is running can omit the user name and password and still be able call any of the DTI functions and view or modify all configuration settings.

However, if the DTI application is running locally through a Terminal Services session or running remotely, provide the user name and password of a user with administrative level privileges on the server machine. This ensures that the application has full access for the DTI functions. Without administrator level privileges, an application returns an access error for most of the DTI functions. Only a subset of the functions work. For example, many of the functions that can modify configuration settings when full access is permitted are restricted to read-only access.

Note: You must call PvStart() to initialize DTI before attempting to connect to a server using this function.

Example

```
BTI_CHAR_PTR uName = "jsmith";
BTI_CHAR_PTR pword = "123";
BTI_CHAR_PTR svrName = "myserver";
BTI_LONG_PTR phConn = 0xFFFFFFFF;
BTI_SINT status = 0;

status = PvConnectServer(svrName,
                         uName,
                         pword,
                         &phConn);
```

See Also

PvStart()
PvGetServerName()
PvDisconnect()
PvStop()

PvCopyDatabase()

Copies a database to a new database, adjusting the referential integrity if needed.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvCopyDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR_PTR      newdbName,
   BTI_CHAR_PTR      newdictPath,
   BTI_CHAR_PTR      newdataPath);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database to copy.
In	*newdbName*	Name of the new database.
In	*newdictPath*	Dictionary path of the new database.
In	*newdataPath*	Data path. Pass an empty string to use the default data path (that is, the same as the dictionary path) If you want to create a new database that consists of MicroKernel Engine data files located in multiple paths, specify this parameter as a semicolon (;) delimited list. For example: `C:\data\path1;C:\data\path2`

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_ACCESS_RIGHT	Insufficient access right for the operation
P_E_DICTIONARY_ALREADY_EXISTS	Cannot create dictionary because it already exists.

P_E_SHARED_DDF_EXIST	The dictionary path is being used by another database.
P_E_DUPLICATE_NAME	Named database already exists on the server.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- The database and its files must be closed.
- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Example

```
BTI_LONG connectionHandle = P_LOCAL_DB_CONNECTION;
BTI_CHAR_PTR newdataPath = "c:\\data\\gallery2";
BTI_CHAR_PTR newdictPath = "c:\\data\\gallery2";
BTI_CHAR_PTR databaseName = "Gallery";
BTI_CHAR_PTR newdatabaseName = "GalleryCopy";
BTI_SINT status = 0;
BTI_CHAR_PTR server = "MyServer";
BTI_CHAR_PTR user = "Administrator";
BTI_CHAR_PTR pwd = "Admin";
//only need to connect to server if it is remote
//otherwise can pass P_LOCAL_DB_CONNECTION for the handle

status = PvCopyDatabase(
connectionHandle,
databaseName,
newdatabaseName
dictPath,
dataPath);
```

See Also

PvStart()
PvConnectServer()
PvCreateDatabase()
PvGetDbFlags()
PvModifyDatabase()
PvDropDatabase()
PvDisconnect()
PvStop()

PvCountDSNs()

Retrieves the number of datasource names (DSN).

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvCountDSNs(
   BTI_LONG          hConnection,
   BTI_ULONG_PTR     pdsnCount,
   BTI_CHAR          filtering);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pdsnCount*	Address of an unsigned long to receive the number of DSNs.
In	*filtering*	Set to 1 if you want only Zen DSNs. Set to 0 if you want all DSNs.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

To retrieve the number of DSNs without having to prompt the user to login, pass empty strings for username and password when establishing the server connection with PvConnectServer().

Note: The connection established by passing empty strings for username and password is an insecure connection, and will not have sufficient rights to perform most of the other operations in DTI.

See Also

PvStart()
PvConnectServer()
PvListDSNs()
PvGetDSN()
PvDisconnect()
PvStop()

PvCountSelectionItems()

Count the number of selection items for a setting of types (PVSETTING_SINGLE_SEL or PVSETTING_MULTI_SEL).

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvCountSelectionItems(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pNumItems);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of categories can be obtained with the PvGetCategoryList() function. A list of settings for a particular category can be obtained from PvGetSettingList().
Out	*pNumItems*	Address of an unsigned long that receives the number of selection items.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_INVALID_DATA_TYPE	The requested setting is not of selection type.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvDisconnect()
PvStop()

PvCreateDatabase()

Creates a database by adding an entry to dbnames.cfg file. This entry is later used to create DSNs.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvCreateDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR_PTR      dictPath,
   BTI_CHAR_PTR      dataPath,
   BTI_ULONG         dbFlags);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the **PvConnectServer()** function.
In	*dbName*	Name of the database.
In	*dictPath*	Dictionary path.
In	*dataPath*	Data path. Pass an empty string to use the default data path (that is, the same as the dictionary path) If you want to create a database that consists of MicroKernel Engine data files located in multiple paths, specify this parameter as a semicolon (;) delimited list. For example: `C:\data\path1;C:\data\path2`

In	*dbFlags*	Database flags, which can be a combination of the P_DBFLAG_ constants. • P_DBFLAG_RI (enforce integrity constraints, including referential integrity and triggers) • P_DBFLAG_BOUND. Create DDF files and stamp the database name on the dictionary files so only that the database can use them. If the database is not bound, then more than one database can use the same dictionary file set. If you are creating a bound database and want to bind to DDF files that already exist, specify both P_DBFLAG_CREATE_DDF and P_DBFLAG_BOUND. • P_DBFLAG_CREATE_DDF (create DDF files. The directory specified for *dictPath* has to exist.) • P_DBFLAG_DBSEC_AUTHENTICATION (use database security authentication, Mixed security policy. See Btrieve Security Policy.) • P_DBFLAG_DBSEC_AUTHORIZATION (use database security authorization, Database security policy. See Btrieve Security Policy.) • P_DBFLAG_LONGMETADATA (use V2 metadata. See Metadata Version.)

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_ACCESS_RIGHT	Insufficient access right for the operation
P_E_DICTIONARY_ALREADY_EXISTS	Cannot create dictionary because it already exists.
P_E_SHARED_DDF_EXIST	The dictionary path is being used by another database.
P_E_DUPLICATE_NAME	Named database already exists on the server.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Btrieve Security Policy

The following table indicates how to specify a security model in a new database, or to interpret the security model of an existing database. Using any other combination of flags for security will result in status code 7024.

This Flag Combination	Represents This Security Model
No flags	Classic
P_DBFLAG_DBSEC_AUTHENTICATION	Mixed
P_DBFLAG_DBSEC_AUTHENTICATION P_DBFLAG_DBSEC_AUTHORIZATION	Database

Metadata Version

If you specify P_DBFLAG_LONGMETADATA, then the database property in dbnames.cfg is set to V2 metadata. If you specify both P_DBFLAG_LONGMETADATA and P_DBFLAG_CREATE_DDF, then the DDFs created are also V2 metadata.

The result of DDF creation varies depending on the DDF versions that already exist in the dictionary location.

Dictionary Location Contains	Result of DDF Creation
No DDFs	New DDFs added to dictionary location
DDFs of other metadata version	New DDFs added to group of existing DDFs
DDFs of same metadata version	New DDFs overwrite existing DDFs. Information in old DDFs is lost.

For example, suppose that your dictionary location contains V1 metadata DDFs and you create V2 metadata DDFs. The dictionary location will then contain a combination of V1 metadata DDFs and V2 metadata DDFs. A particular database can use one set of DDFs or the other, but not both concurrently.

Example

The following example creates a database and DDFs that uses V2 metadata.

```
BTI_LONG connectionHandle = P_LOCAL_DB_CONNECTION;
BTI_CHAR_PTR dataPath = "c:\\data\\gallery";
BTI_CHAR_PTR dictPath = "c:\\data\\gallery";
BTI_CHAR_PTR databaseName = "Gallery";
```

```
BTI_SINT status = 0;
BTI_CHAR_PTR server = "MyServer";
BTI_CHAR_PTR user = "Administrator";
BTI_CHAR_PTR pwd = "Admin";
//only need to connect to server if it is remote
//otherwise can pass P_LOCAL_DB_CONNECTION for the handle

status = PvCreateDatabase(
connectionHandle,
databaseName,
dictPath,
dataPath,
P_DBFLAG_CREATE_DDF,
P_DBFLAG_LONGMETADATA);
```

See Also

PvStart()
PvConnectServer()
PvGetDbFlags()
PvModifyDatabase()
PvDropDatabase()
PvDisconnect()
PvStop()

PvCreateDatabase2()

Creates a database by adding an entry to dbnames.cfg file. This function is the same as PvCreateDatabase() except that the database code page is also specified.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvCreateDatabase2(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR_PTR      dictPath,
   BTI_CHAR_PTR      dataPath,
   BTI_ULONG         dbFlags,
   BTI_LONG          dbCodePage);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the **PvConnectServer()** function.
In	*dbName*	Name of the database.
In	*dictPath*	Dictionary path.
In	*dataPath*	Data path. Pass an empty string to use the default data path (that is, the same as the dictionary path) If you want to create a database that consists of MicroKernel Engine data files located in multiple paths, specify this parameter as a semicolon (;) delimited list. For example: C:\data\path1;C:\data\path2

In	*dbFlags*	Database flags, which can be a combination of the P_DBFLAG_ constants. • P_DBFLAG_RI (enforce integrity constraints, including referential integrity and triggers) • P_DBFLAG_BOUND (create DDF files and stamp the database name on the dictionary files so only that database can use them. If the database is not bound, then several databases can use the same dictionary file set.) If trying to create a bound database and you want to bind to DDF files that already exist, specify both P_DBFLAG_CREATE_DDF and P_DBFLAG_BOUND. • P_DBFLAG_CREATE_DDF (create DDF files. The directory specified for *dictPath* has to exist.) • P_DBFLAG_DBSEC_AUTHENTICATION (use database security authentication, Mixed security policy. See Btrieve Security Policy.) • P_DBFLAG_DBSEC_AUTHORIZATION (use database security authorization, Database security policy. See Btrieve Security Policy.) • P_DBFLAG_LONGMETADATA (use V2 metadata. See Metadata Version.)
In	*dbCodePage*	For databases on Windows platforms, a number indicating the code page for database data and metadata strings. For databases on Linux and macOS distributions, one of the following to indicate the code page for database data and metadata strings: • P_DBCODEPAGE_UTF8 • P_DBCODEPAGE_EUCJP • P_DBCODEPAGE_ISO8859_1 For databases on Windows, Linux, and macOS, a value of zero can also be used. Zero indicates legacy behavior. That is, no code page is specified, defaulting to the operating system encoding on the server machine. See also the Code Page database property in *Zen User's Guide*. **Note:** The database engine does **not** validate the encoding of the data and metadata that an application inserts into a database. The engine assumes that all data was entered using the encoding of the server or the client as explained under Database Code Page and Client Encoding in *Advanced Operations Guide*.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.

P_E_NULL_PTR	Call with NULL pointer
P_E_ACCESS_RIGHT	Insufficient access right for the operation
P_E_DICTIONARY_ALREADY_EXISTS	Cannot create dictionary because it already exists.
P_E_SHARED_DDF_EXIST	The dictionary path is being used by another database.
P_E_DUPLICATE_NAME	Named database already exists on the server.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Btrieve Security Policy and Metadata Version

See Btrieve Security Policy and Metadata Version, respectively.

See Also

PvConnectServer()
PvCreateDSN2()
PvDisconnect()
PvDropDatabase()
PvGetDbCodePage()
PvGetDbFlags()
PvGetDSNEx2()
PvModifyDatabase2()
PvStart()
PvStop()

PvCreateDictionary()

Creates a new set of dictionary files. Given a fully-qualified path for the dictionary, it returns a dictionary handle that will be used for any subsequent calls to catalog functions.

Note: This function is deprecated in Zen 9 and higher versions. See PvCreateDatabase() and PvOpenDatabase() to replace this function in your application.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvCreateDictionary(
   LPCSTR          path,
   WORD*           dictHandle,
   LPCSTR          user,
   LPCSTR          password);
```

Arguments

In	*path*	Fully-qualified path to the dictionary files.
Out	*dictHandle*	Handle to be used in subsequent calls
In	*user*	User name used with the new dictionary. This argument can be set to NULL.
In	*password*	Used in conjunction with user name to create new dictionary files. Can also be NULL.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errPathNotFound	The specified path is invalid.
PCM_errSessionSecurityError	Either the user name or password is invalid.
PCM_errDictionaryAlreadyExists	A set of ddf files already exists at the specified location.

Remarks

Use PvCloseDictionary() to free the resources.

See Also

PvStart()
PvOpenDatabase()
PvGetDbDictionaryPath()
PvCloseDictionary()
PvStop()

PvCreateDSN()

Creates a new engine data source name (DSN).

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to create client DSNs (or dsnadd utility on Linux and macOS).

Syntax

```
BTI_API PvCreateDSN(
   BTI_LONG       hConnection,
   BTI_CHAR_PTR   pdsnName,
   BTI_CHAR_PTR   pdsnDesc,
   BTI_CHAR_PTR   pdsnDBQ,
   BTI_LONG       openMode);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pdsnName*	Name for the new DSN.
In	*pdsnDesc*	Description for the new DSN.
In	*pdsnDBQ*	Database name to which this DSN will connect. This name must already exist. To create a database name, see PvCreateDatabase().
In	*OpenMode*	Open mode for the DSN, which is one of the following: • NORMAL_MODE • ACCELERATED_MODE • READONLY_MODE • EXCLUSIVE_MODE

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_INVALID_NAME	The specified DSN name is invalid.
P_E_DSN_ALREADY_EXIST	The specified DSN name already exists.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_INVALID_OPEN_MODE	The specified open mode is invalid.
P_E_FAIL	Failed to retrieve data path.

Remarks

This function creates engine DSNs only. To create a client DSN, you must use the ODBC API.

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- The database name referenced in the *pdsnDBQ* parameter must already exist. To create a database name, see PvCreateDatabase().

See Also

PvStart()
PvConnectServer()
PvListDSNs()
PvModifyDSN()
PvGetDSN()
PvGetDSNEx()
PvDeleteDSN()
PvCountDSNs()
PvStop()

PvCreateDSN2()

Creates a new engine data source name (DSN) and specifies the encoding option for data.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to create client DSNs (or **dsnadd** utility on Linux and macOS).

Syntax

```
BTI_API PvCreateDSN2(
   BTI_LONG       hConnection,
   BTI_CHAR_PTR   pdsnName,
   BTI_CHAR_PTR   pdsnDesc,
   BTI_CHAR_PTR   pdsnDBQ,
   BTI_LONG       openMode,
   BTI_LONG       translate);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pdsnName*	Name for the new DSN.
In	*pdsnDesc*	Description for the new DSN.
In	*dsnDBQ*	Database name to which this DSN will connect. This name must already exist. To create a database name, see PvCreateDatabase().
In	*OpenMode*	Open mode for the DSN, which is one of the following: • NORMAL_MODE • ACCELERATED_MODE • READONLY_MODE • EXCLUSIVE_MODE See also DSN Open Mode in *ODBC Guide*.

In	*translate*	Encoding option for data, which can be one of the following: • DSNFLAG_DEFAULT • DSNFLAG_OEMANSI • DSNFLAG_AUTO See also Encoding Translation in *ODBC Guide*. Note that DSNFLAG_DEFAULT corresponds to the "None" encoding option in ODBC Administrator.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_INVALID_NAME	The specified DSN name is invalid.
P_E_DSN_ALREADY_EXIST	The specified DSN name already exists.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_INVALID_OPEN_MODE	The specified open mode is invalid.
P_E_INVALID_TRANSLATE_OPTION	The specified encoding translation option is invalid.
P_E_FAIL	Failed to retrieve data path.

Remarks

This function creates engine DSNs only and requires a PSQL v10 client or later. To create a client DSN, you must use the ODBC API. (On Linux and macOS, you can also use the dsnadd utility to create a client DSN.)

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- The database name referenced in the *pdsnDBQ* parameter must already exist. To create a database name, see PvCreateDatabase().

See Also

PvStart()
PvConnectServer()
PvListDSNs()
PvModifyDSN()
PvGetDSN()
PvGetDSNEx()
PvDeleteDSN()
PvCountDSNs()
PvStop()

PvCreateGroup()

Creates a new user group in the existing database.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvCreateGroup(
   BTI_WORD            dbHandle,
   const BTI_CHAR*     group);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*Group*	Database group name.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The specified group name is invalid.
PCM_errDatabaseHasNoSecurity	Database has no security
PCM_errSessionSecurityError	Database opened with insufficient privilege
PCM_errGroupAlreadyExists	Group already exists

Remarks

The following preconditions must be met:

- You must first open a database successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.
- A group with the same name cannot already exist in the specified database.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvAddUserToGroup()
PvRemoveUserFromGroup()
PvCreateUser()
PvAlterUserName()
PvAlterUserPassword()
PvDropGroup()
PvDropUser()
PvOpenDatabase()
PvCloseDatabase()

PvCreateUser()

Creates a new user in the existing database. Optionally set a password and assign the new user to an existing group.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvCreateUser(
   BTI_WORD            dbHandle,
   const BTI_CHAR*     user,
   const BTI_CHAR*     password,
   const BTI_CHAR*     group);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*user*	Database user name.
In	*password*	User password. If set to NULL, no password is set.
In	*group*	Database group name for user. If set to NULL, user is not assigned to a group.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The specified account or user name is invalid.
PCM_errUserAlreadyExists	User already exists.
PCM_errDatabaseHasNoSecurity	Database has no security.
PCM_errSessionSecurityError	Database opened with insufficient privilege.

Remarks

The following preconditions must be met:

- You must first open a database successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.
- A user with the same name cannot already exist in the specified database.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvAlterUserName()
PvAlterUserPassword()
PvAddUserToGroup()
PvRemoveUserFromGroup()
PvCreateGroup()
PvDropUser()
PvOpenDatabase()
PvCloseDatabase()

PvDeleteDSN()

Deletes a data source name.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to work with client DSNs.

Syntax

```
BTI_API PvDeleteDSN(
   BTI_LONG       hConnection,
   BTI_CHAR_PTR   pdsnName);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pdsnName*	DSN to delete.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_DSN_DOES_NOT_EXIST	The specified DSN name does not exist.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_FAIL	Failed to retrieve data path.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart()

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvListDSNs()
PvModifyDSN()
PvGetDSN()
PvGetDSNEx()
PvCreateDSN()
PvCountDSNs()
PvStop()

PvDeleteLicense()

Deletes (deauthorizes) the specified license from the computer indicated by the connection.

Header File: dtilicense.h (See also Header Files)

Function First Available In Library: w3dbav80.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvDeleteLicense(
   BTI_LONG        hConnection,
   BTI_CHAR_PTR    licenses);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*licenses*	License to be deleted.

Return Values

P_OK	The operation completed successfully
P_E_FAIL	The operation did not complete successfully
P_E_LIC_NOT_FOUND	The license specified is not currently authorized.
P_E_LIC_INVALID	The license specified is invalid.
Status code pertaining to license administration or to authorization	See *Status Codes and Messages* for License Administrator Status Codes and Authorization Status Codes.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Example

```
BTI_CHAR_PTR delete_lic = "ERXVD3U4ZS9KR94QPDHV5BN2";
status = PvDeleteLicense(P_LOCAL_DB_CONNECTION, delete_lic);
```

See Also

PvAddLicense()
PvValidateLicenses()
PvGetProductsInfo()
PvStop()
PvStart()

PvDisconnect()

Attempts to disconnect the connection established earlier by PvConnectServer function.

Header File: connect.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvDisconnect(
   BTI_LONG          hConnection);
```

Arguments

In	*hConnection*	Connection handle to be disconnected.Connection handles are obtained with the PvConnectServer() function.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_FAIL	Failed to disconnect to the named server.

Example

```
BTI_SINT status = 0;
status = PvDisconnect(m_hConn);
```

See Also

PvStart()
PvConnectServer()
PvGetMkdeClientsData()
PvGetMkdeCommStat()
PvGetMkdeUsage()
PvGetOpenFilesData()
PvFreeOpenFilesData()

PvDisconnectMkdeClient()
PvDisconnectSQLConnection()
PvStop()

PvDisconnectMkdeClient()

Attempts to disconnect an active MicroKernel Engine client by specifying a client ID. In order to obtain a valid client ID, use PvGetMkdeClientData and PvGetMkdeClientId functions.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvDisconnectMkdeClient(
   BTI_LONG            hConnection,
   PVCLIENTID*         pClientId);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pClientId*	Address of the PVCLIENTID structure to identify the MicroKernel Engine client.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_CLIENT	Invalid client ID.
P_E_FAIL	Failed to disconnect to the named server.

Example

```
unsigned long count = 0;

// This sample disconnects all active Mkde connections
BTI_SINT status = 0
PVCLIENTID clientId;
 status = PvGetMkdeClientsData(connection, &count);
```

```
while (count > 0)
{
status = PvGetMkdeClientId(connection, 0, &client Id);
status = PvDisconnectMkdeClient(connection, &clientId);
status = PvGetMkdeClientsData(connection, &count)
}
  PvFreeMkdeClientsData(connection);
```

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetMkdeClientsData()
PvGetMkdeClientId()
PvGetMkdeClientInfo()
PvGetMkdeClientHandlesData()
PvDisconnect()
PvStop()

PvDisconnectSQLConnection()

Attempts to disconnect an active SQL connection by passing SQL connection Id. Use PvGetSQLConnectionsData and PvSQLConnectionInfo to obtain a valid connection Id.

Note: Each SQL connection also establishes a MicroKernel Engine connection. Use PvDisconnectMKDEClient to kill those connections.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvDisconnectSQLConnection(
   BTI_LONG            hConnection,
   PVSQLCONNID*        pSQLConnId);
```

Arguments

In	*hConnection*	Server connection handle that contains the SQL connection to be disconnected. Server connection handles are obtained with the PvConnectServer() function.
In	*pSQLConnId*	Address of the PVSQLCONNID structure to identify the SQL connection. SQL connections are obtained with the PvGetSQLConnectionsData()

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_CLIENT	Invalid client ID.
P_E_FAIL	Failed to disconnect to the named server.

Example

```
BTI_SINT status = 0;
```

```
PVSQLCONNINFO connectionInfo;
PVSQLCONNID   connId;
status = PvGetSQLConnectionsData (connection, &count);
while (count > 0)
 {
     status = PvGetSQLConnectionInfo(connection, 0,
                  &connectionInfo);
      connId.u132ProcessId =
           connectionInfo.u132ProcessId;
      connId.u132ThreadId =
           connectionInfo.u132ThreadId;
      status = PvDisconnectSQLConnection(connection,
                  &connId);
      status = PvGetSQLConnectionsData (connection,
                  &count);
      }
     PvFreeSQLConnectionsData(connection, &count);
```

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetSQLConnectionsData()
PvGetSQLConnectionInfo()
PvDisconnect()
PvStop()

PvDropDatabase()

Deletes a specified entry from dnames.cfg.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvDropDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR          option);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database. A list of all database names for a particular server is obtained with the PvGetDbNamesData() function. A single database name from the resulting list can be obtained with the PvGetDbName() function.
In	*option*	Bit mask that specifies options. Set the low-order bit to one (0001h) if you want DDF files to be deleted in addition to the database name. Otherwise, only the database name will be deleted but DDF files will remain.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_ACCESS_RIGHT	Insufficient access right for the operation
P_E_NOT_EXIST	Named database does not exist.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvCreateDatabase()
PvModifyDatabase()
PvDisconnect()
PvStop()

PvDropGroup()

Drop an existing group from the database.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvDropGroup(
   BTI_WORD            dbHandle,
   const BTI_CHAR*     group);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*group*	Database group name.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The specified group name does not exist.
PCM_errDatabaseHasNoSecurity	Database has no security
PCM_errSessionSecurityError	Database opened with insufficient privilege
PCM_errGroupNotEmpty	An user is associated with this group

Remarks

The following preconditions must be met:

- You must first open a database successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.
- The group must already exist in the specified database.
- The group cannot contain any members.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvCreateGroup()
PvAddUserToGroup()
PvRemoveUserFromGroup()
PvDropUser()
PvOpenDatabase()
PvCloseDatabase()

PvDropIndex()

Drops the index from dictionary and data files, given the index number.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvDropIndex(
   WORD             dictHandle,
   LPCSTR           tableName,
   WORD             indexNumber,
   BOOL             renumber);
```

Arguments

In	*dictHandle*	Handle of an open dictionary returned by PvOpenDatabase().
In	*tableName*	Name of the table with the index to be dropped.
In	*indexNumber*	Number of the index to be dropped.
In	*renumber*	Indicates whether the remaining indexes should be renumbered.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidDictionaryHandle	The specified dictionary handle does not exist.
PCM_errTableNotFound	The specified table was not found.
PCM_errInvalidIndex	The specified index was not found.

Remarks

You must first open a dictionary successfully using PvOpenDatabase().

The table specified by *tableName* must exist in the dictionary specified by *dictHandle*.

See Also

PvStart()
PvOpenDatabase()
PvDropIndexByName()
PvAddIndex()
PvCloseDictionary()
PvStop()

PvDropIndexByName()

Drops the index from dictionary and data files, given a name.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvDropIndexByName(
   WORD            dictHandle,
   LPCSTR          tableName,
   LPCSTR          indexName);
```

Arguments

In	*dictHandle*	Handle of an open dictionary returned by PvOpenDatabase().
In	*tableName*	Name of the table with the index to be dropped.
In	*indexName*	Name of the index to be dropped.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidDictionaryHandle	The specified dictionary handle does not exist.
PCM_errTableNotFound	The table specified in *tableName* was not found in the dictionary.

Remarks

You must first open a dictionary successfully using PvOpenDatabase().

The table specified by *tableName* must exist in the dictionary specified by *dictHandle*.

See Also

PvStart()
PvOpenDatabase()
PvAddIndex()
PvDropIndex()
PvCloseDictionary()
PvStop()

PvDropTable()

Drops the specified table from the open dictionary specified by the dictionary handle.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvDropTable(
   WORD              dictHandle,
   LPCSTR            tableName,
   WORD              keepFile);
```

Arguments

In	*dictHandle*	Handle of an open dictionary returned by PvOpenDatabase().
In	*tableName*	Name of the table to delete.
In	*keepFile*	Indicates whether or not the data file will be deleted. If set to 0, the data file associated with the table will be deleted. If non-zero, the data file will not be deleted.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidDictionaryHandle	The specified dictionary handle does not exist.
PCM_errTableNotFound	The specified table name was not found.

Remarks

You must first open a dictionary successfully using PvOpenDatabase().

The table specified by *tableName* must exist in the dictionary specified by *dictHandle*.

See Also

PvStart()
PvOpenDatabase()
PvGetTableNames()
PvGetTable()
PvAddTable()
PvCloseDictionary()
PvStop()

PvDropUser()

Drop an existing user from the database.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvDropUser (
   BTI_WORD           dbHandle,
   const BTI_CHAR*    user);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*user*	Database user name

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The specified account or user name does not exist.
PCM_errNotAllowedToDropAdministrator	Attempt to drop Master user.
PCM_errDatabaseHasNoSecurity	Database has no security.
PCM_errSessionSecurityError	Database opened with insufficient privilege.

Remarks

The following preconditions must be met:

- You must first open a database successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.
- A user with the same name must already exist in the specified database.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvAddUserToGroup()
PvAlterUserName()
PvAlterUserPassword()
PvCreateUser()
PvRemoveUserFromGroup()
PvOpenDatabase()
PvCloseDatabase()

PvFreeDbNamesData()

Free the resource allocated for database names on a connected server. This function needs to be called after preceding calls to PvGetDbNamesData.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvFreeDbNamesData(
   BTI_LONG            hConnection);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to database names not available.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Database names data retrieved by calling PvGetDbNamesData().

See Also

PvStart()
PvConnectServer()
PvGetDbNamesData()
PvGetDbName()
PvDisconnect()
PvStop()

PvFreeMkdeClientsData()

Free the cached information related to the active MicroKernel Engine clients. This function needs to be called after preceding calls to PvGetMkdeClientsData.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvFreeMkdeClientsData(
   BTI_LONG            hConnection);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for active clients retrieved by calling PvGetMkdeClientsData();

See Also

PvStart()
PvConnectServer()
PvGetMkdeClientsData()
PvGetMkdeClientInfo()
PvDisconnect()
PvStop()

PvFreeOpenFilesData()

Free the cached information related to the open files. This function needs to be called after preceding calls to PvGetOpenFilesData.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvFreeOpenFilesData(
   BTI_LONG            hConnection);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetOpenFilesData().

See Also

PvStart()
PvConnectServer()
PvGetOpenFilesData()
PvGetOpenFileName()
PvDisconnect()
PvStop()

PvFreeSQLConnectionsData()

Free the cached information related to SQL connections. This function needs to be called after preceding calls to PvGetSQLConnectionsData.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvFreeSQLConnectionsData(
   BTI_LONG            hConnection);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetSQLConnectionsData().

See Also

PvStart()
PvConnectServer()
PvGetSQLConnectionsData()
PvGetSQLConnectionInfo()
PvDisconnect()
PvStop()

PvFreeTable()

Frees memory allocated by a PvGetTable() function call.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvFreeTable(
   TABLEINFO*      tableProps,
   COLUMNMAP*      columnList,
   INDEXMAP*       indexList);
```

Arguments

In/Out	*tableProps*	Pointer to a structure containing table information
In/Out	*columnList*	Pointer to an array of columns defined in the table.
In/Out	*indexList*	Pointer to an array of segments defined in the table.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	A general failure occurred

Remarks

This function frees the structures created during a PvGetTable() call.

Example

```
PRESULT status = 0;
status = PvFreeTable(mytableProps, MyColumnList MyindexList);
```

See Also

PvStart()
PvOpenDatabase()
PvGetTableNames()
PvGetTable()
PvFreeTableNames()
PvCloseDictionary()
PvStop()

PvFreeTableNames()

Frees memory allocated with a PvGetTableNames() call.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvFreeTableNames(
   TABLEMAP*       tableList);
```

Arguments

In/Out	*tableList*	Array of TABLEMAP structures that contain table names.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.

Remarks

The memory freed with this function is successfully allocated during a PvGetTableNames() call to retrieve all of the table names for a specified dictionary.

Example

```
PRESULT status = 0;
status = PvFreeTableNames(&mytableList);
```

See Also

PvStart()
PvOpenDatabase()
PvGetTableNames()
PvGetTable()

PvFreeTable()
PvCloseDictionary()
PvStop()

PvGetAllPossibleSelections()

Retrieves all available selection choices for a setting of types (PVSETTING_SINGLE_SEL or PVSETTING_MULTI_SEL).

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetAllPossibleSelections(
   BTI_LONG            hConnection,
   BTI_ULONG           settingID,
   BTI_ULONG_PTR       pNumItems,
   BTI_ULONG_PTR       pSelectionList);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pNumItems*	Address of an unsigned long that receives the total number of selection items. You can also retrieve the number of selection items by calling PvCountSelectionItems()
Out	*pSelectionList*	Array that contains all available selection choices.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_INVALID_DATA_TYPE	The requested setting is not of selection type.
P_E_BUFFER_TOO_SMALL	The array size is too small. In this case, the required size is returned in *pNumItems*.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvCountSelectionItems()
PvDisconnect()
PvStop()

PvGetBooleanStrings()

Retrieves display string related to Boolean type setting.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetBooleanStrings(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_LONG_PTR      trueStringSize,
   BTI_CHAR_PTR      trueString,
   BTI_LONG_PTR      falseStringSize,
   BTI_CHAR_PTR      falseString);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
Out	*trueStringSize*	Long integer containing the length of *trueString*.
Out	*trueString*	Display string for TRUE (size >= 16 bytes).
Out	*falseStringSize*	Long integer containing the length of *falseString*.
Out	*falseString*	Display string for FALSE (size >= 16 bytes).

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The requested setting is not of long type.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvDisconnect()
PvStop()

PvGetBooleanValue()

Retrieves the value for a Boolean type setting. Either default or current value can be retrieved.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetBooleanValue(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_SINT_PTR      pValue,
   BTI_SINT          whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
Out	*pValue*	Address of a Boolean variable that receives the setting value.
In	*whichData*	Flag to indicate which value is requested: PVDATA_DEFAULT returns default value. PVDATA_CURRENT returns current value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_INVALID_DATA_TYPE	The requested setting is not of Boolean type.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetBooleanStrings()
PvSetBooleanValue()
PvDisconnect()
PvStop()

PvGetCategoryInfo()

Retrieves information about a category of engine settings.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetCategoryInfo(
   BTI_LONG          hConnection,
   BTI_ULONG         categoryID,
   PVCATEGORYINFO*   pCatInfo);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*categoryID*	Unique identifier for the category. You can obtain a list of identifiers via the PvGetCategoryList() function.
Out	*pCatInfo*	Address of a PVCATEGORYINFO structure that will receive the category information.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

The number of settings returned in the PVCATEGORYINFO structure represents the total number of settings for that category, both client and server. To get the applicable number of settings, call PvGetSettingList(). If it is a remote connection, the server side settings are not applicable.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvDisconnect()
PvStop()

PvGetCategoryList()

Retrieves the list of category IDs on the engine specified by the current connection.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetCategoryList(
   BTI_LONG          hConnection,
   BTI_ULONG_PTR     pnumCategories,
   BTI_ULONG_PTR     pCategoriesList);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In/Out	*pnumCategories*	Address of an unsigned long containing the number of categories that can be returned in *pCategoriesList*. You can also call PvGetCategoryListCount() to retrieve this value.
Out	*pCategoriesList*	Array containing the category IDs.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_FAIL	Failed for other reasons.
P_E_BUFFER_TOO_SMALL	Array size is too small. The required size is returned in *pnumCategories*.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryInfo()
PvDisconnect()
PvStop()

PvGetCategoryListCount()

Retrieves the number of categories on the engine specified by the current connection. This number can then be used to allocate an array to pass to PvGetCategoryList().

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetCategoryListCount(
   BTI_LONG            hConnection,
   BTI_ULONG_PTR       pListCount);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pListCount*	Address of an unsigned long containing the number of categories.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryInfo()
PvDisconnect()
PvStop()

PvGetDbCodePage()

Retrieves the code page associated with a named database.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetDbCodePage(
   BTI_LONG       hConnection,
   BTI_CHAR_PTR   dbName,
   BTI_LONG_PTR   pDbCodePage);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database. A list of all database names for a particular server is obtained with the PvGetDbNamesData() function. A single database name from the resulting list can be obtained with the PvGetDbName() function.
Out	*pDbCodePage*	Code page of the database. A value of zero indicates the default code page on the server.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_NOT_EXIST	Named database does not exist.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvConnectServer()
PvCreateDatabase2()
PvCreateDSN2()
PvModifyDatabase2()
PvModifyDSN2()
PvGetDSNEx2()
PvStart()

PvGetDbDataPath()

Retrieves the data path (where data files reside) of a named database. This information is stored in dbnames.cfg.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetDbDataPath(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      dataPath);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database. A list of all database names for a particular server is obtained with the PvGetDbNamesData() function. A single database name from the resulting list can be obtained with the PvGetDbName() function.
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer. Receives actual size of the path returned.
Out	*dataPath*	Contains the data path if successful, or empty string otherwise.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in *pBufSize*.
P_E_NOT_EXIST	Named database does not exist.

P_E_FAIL	Failed for other reasons.

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetDbNamesData()
PvGetDbName()
PvGetDbDictionaryPath()
PvGetDbServerName()
PvDisconnect()
PvStop()

PvGetDbDictionaryPath()

Retrieves the dictionary path (where DDF files reside) of a named database.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetDbDictionaryPath(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      dictPath);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database. A list of all database names for a particular server is obtained with the PvGetDbNamesData() function. A single database name from the resulting list can be obtained with the PvGetDbName() function.
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer. Receives actual size of the path returned.
Out	*dictPath*	Contains the dictionary path if successful, or empty string otherwise.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in *pBufSize*.
P_E_NOT_EXIST	Named database does not exist.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetDbNamesData()
PvGetDbName()
PvGetDbDataPath()
PvGetDbServerName()
PvDisconnect()
PvStop()

PvGetDbFlags()

Retrieves the database flags associated with a named database.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetDbFlags(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_ULONG_PTR     pDbFlags);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database. A list of all database names for a particular server is obtained with the PvGetDbNamesData() function. A single database name from the resulting list can be obtained with the PvGetDbName() function.
Out	*pDbFlags*	Database flags, which can be a combination of the P_DBFLAG_ constants. • P_DBFLAG_RI (integrity constraints, including referential integrity and triggers) • P_DBFLAG_BOUND (DDF files stamped with the database name so only that database can use them) • P_DBFLAG_DBSEC_AUTHENTICATION (Mixed security policy. See Btrieve Security Policy.) • P_DBFLAG_DBSEC_AUTHORIZATION (Database security policy. See Btrieve Security Policy.) • P_DBFLAG_LONGMETADATA (see Metadata Version)

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer

P_E_NOT_EXIST	Named database does not exist.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Btrieve Security Policy

The following table indicates how to interpret the security model of an existing database.

This Flag Combination	Represent This Security Model
No flags	Classic
P_DBFLAG_DBSEC_AUTHENTICATION	Mixed
P_DBFLAG_DBSEC_AUTHENTICATION + P_DBFLAG_DBSEC_AUTHORIZATION	Database

See Also

PvStart()
PvConnectServer()
PvCreateDatabase()
PvModifyDatabase()
PvGetDbNamesData()
PvGetDbName()
PvDisconnect()
PvStop()

PvGetDbName()

Gets the name of a database on a connected server using a sequence number. You can obtain the number of database names by calling the PvGetDbNamesData() function. The sequence number is 1 based.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetDbName(
   BTI_LONG          hConnection,
   BTI_ULONG         sequence,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      dbName);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*sequence*	The sequence number (1 based) of the database name. Must be within a valid range with upper limit defined by PvGetDbNamesData().
In/Out	*pBufSize*	Address of an unsigned long containing size of buffer allocated to receive the database name. Receives actual size of chars copied. The size should include the null terminator.
Out	*dbName*	String value returned.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to database names not available.
P_E_NULL_PTR	Call with NULL pointer
P_E_BUFFER_TOO_SMALL	Allocated buffer is too small for the string.
P_E_INVALID_SEQUENCE	Sequence number is not valid.

P_E_FAIL	Failed for other reasons.

Example

```
BTI_ULONG i;
BTI_ULONG count = 0;
BTI_CHAR dbName[BDB_SIZE_DBNAME+1];
BTI_SINT status = PvGetDbNamesData(connection, &count);
for (i=1; i<= count; i++)
{
   BTI_ULONG dbNameSize = sizeof(dbName);
   status = PvGetDbName(connection, i, &dbNameSize, dbName);
}
status = PvFreeDbNamesData(connection);
```

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Database names data retrieved by calling PvGetDbNamesData()
- Caller has a valid database name sequence number.

See Also

PvStart()
PvConnectServer()
PvGetDbNamesData()
PvFreeDbNamesData()
PvDisconnect()
PvStop()

PvGetDbNamesData()

Retrieves the number of database names for a connected server. Use the PvGetDbName() function to enumerate the names.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetDbNamesData(
   BTI_LONG           hConnection,
   BTI_ULONG_PTR      pCount);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pCount*	Address of an unsigned long to receive the number of database names on the server.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

This function should be called first before calling any other functions to get database names information. The caller should call PvFreeDbNamesData() to free the resources allocated for database names.

See Also

PvStart()
PvConnectServer()
PvGetDbName()
PvFreeDbNamesData()
PvDisconnect()
PvStop()

PvGetDbServerName()

Retrieves the name of the server where the named database resides.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetDbServerName(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      serverName,
   BTI_SINT_PTR      pIsLocal);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database. A list of all database names for a particular server is obtained with the PvGetDbNamesData() function. A single database name from the resulting list can be obtained with the PvGetDbName() function.
In/Out	*pBufSize*	Address of an unsigned long containing the size of the buffer. Actual size of server name is returned.
Out	*serverName*	Contains server name if successful, empty string otherwise.
Out	*pIsLocal*	Returns zero for remote server, non-zero for local server.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in *pBufSize*.
P_E_NOT_EXIST	Named database does not exist.

P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer()or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetDbNamesData()
PvGetDbName()
PvDisconnect()
PvStop()

PvGetDSN()

Retrieves information about the datasource name (DSN).

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to work with client DSNs.

Syntax

```
BTI_API PvGetDSN(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dsnName,
   BTI_ULONG_PTR     pdsnDescSize,
   BTI_CHAR_PTR      dsnDesc,
   BTI_ULONG_PTR     pdsnDBQSize,
   BTI_CHAR_PTR      dsnDBQ);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dsnName*	Name of the datasource. A list of DSNs can be obtained with the PvListDSNs() function.
In/Out	*pdsnDescSize*	Address of an unsigned long containing size of the buffer for DSN description. Receives actual size of DSN description.
Out	*dsnDesc*	Contains the description of DSN if successful.
In/Out	*pdsnDBQSize*	Address of an unsigned long containing size of the buffer for name of database. Receives actual size of database name.
Out	*dsnDBQ*	Contains the name of the database if successful.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.

P_E_NULL_PTR	Call with NULL pointer
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in pdsnDescSize or pdsnDBQSize.
P_E_FAIL	Failed to retrieve data path.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

To retrieve information about a DSN without having to prompt the user to login, pass empty strings for username and password when establishing the server connection with PvConnectServer().

Note: The connection established by passing empty strings for username and password is an insecure connection, and will not have sufficient rights to perform most of the other operations in DTI.

See Also

PvStart()
PvConnectServer()
PvGetDSNEx()
PvListDSNs()
PvCountDSNs()
PvCreateDSN()
PvModifyDSN()
PvDeleteDSN()
PvDisconnect()
PvStop()

PvGetDSNEx()

Retrieves information about the datasource name (DSN). This function is identical to PvGetDSN() except that the DSN open mode is also retrieved.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to work with client DSNs.

Syntax

```
BTI_API PvGetDSNEx(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dsnName,
   BTI_ULONG_PTR     pdsnDescSize,
   BTI_CHAR_PTR      dsnDesc,
   BTI_ULONG_PTR     pdsnDBQSize,
   BTI_CHAR_PTR      dsnDBQ,
   BTI_LONG_PTR      pOpenMode);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dsnName*	Name of the datasource. A list of DSNs can be obtained with the PvListDSNs() function.
In/Out	*pdsnDescSize*	Address of an unsigned long containing size of the buffer for DSN description. Receives actual size of DSN description.
Out	*dsnDesc*	Contains the description of DSN if successful.
In/Out	*pdsnDBQSize*	Address of an unsigned long containing size of the buffer for name of database. Receives actual size of database name.
Out	*dsnDBQ*	Contains the name of the database if successful.

Out	*pOpenMode*	Contains open mode of DSN, which is one of the following: • NORMAL_MODE • ACCELERATED_MODE, • READONLY_MODE • EXCLUSIVE_MODE See also DSN Open Mode in *ODBC Guide*.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in pdsnDescSize or pdsnDBQSize.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_DSN_DOES_NOT_EXIST	The specified DSN does not exist.
P_E_INVALID_OPEN_MODE	Invalid open mode.
P_E_FAIL	Failed to retrieve data path.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

To retrieve information about a DSN without having to prompt the user to login, pass empty strings for username and password when establishing the server connection with PvConnectServer().

Note: The connection established by passing empty strings for username and password is an insecure connection, and will not have sufficient rights to perform most of the other operations in DTI.

See Also

PvStart()
PvConnectServer()
PvListDSNs()
PvCountDSNs()
PvGetDSN()
PvCreateDSN()
PvModifyDSN()
PvDeleteDSN()
PvDisconnect()
PvStop()

PvGetDSNEx2()

Retrieves information about the data source name (DSN). This function is the same as PvGetDSNEx() except that the encoding option for data is also retrieved.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to work with client DSNs.

Syntax

```
BTI_API PvGetDSNEx2(
   BTI_LONG           hConnection,
   BTI_CHAR_PTR       dsnName,
   BTI_ULONG_PTR      pdsnDescSize,
   BTI_CHAR_PTR       dsnDesc,
   BTI_ULONG_PTR      pdsnDBQSize,
   BTI_CHAR_PTR       dsnDBQ,
   BTI_LONG_PTR       pOpenMode,
   BTI_LONG_PTR       translate);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dsnName*	Name of the datasource. A list of DSNs can be obtained with the PvListDSNs() function.
In/Out	*pdsnDescSize*	Address of an unsigned long containing size of the buffer for DSN description. Receives actual size of DSN description.
Out	*dsnDesc*	Contains the description of DSN if successful.
In/Out	*pdsnDBQSize*	Address of an unsigned long containing size of the buffer for name of database. Receives actual size of database name.
Out	*dsnDBQ*	Contains the name of the database if successful.

Out	*pOpenMode*	Open mode for the DSN, which is one of the following: • NORMAL_MODE • ACCELERATED_MODE • READONLY_MODE • EXCLUSIVE_MODE See also DSN Open Mode in *ODBC Guide*.
Out	*translate*	Encoding option for data, which can be one of the following: • DSNFLAG_DEFAULT • DSNFLAG_OEMANSI • DSNFLAG_AUTO See also DSN Open Mode in *ODBC Guide*. Note that DSNFLAG_DEFAULT corresponds to the "None" encoding option in ODBC Administrator.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in pdsnDescSize or pdsnDBQSize.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_DSN_DOES_NOT_EXIST	The specified DSN does not exist.
P_E_INVALID_OPEN_MODE	Invalid open mode.
P_E_INVALID_TRANSLATE_OPTION	The specified encoding translation option is invalid.
P_E_FAIL	Failed to retrieve data path.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

To retrieve information about a DSN without having to prompt the user to login, pass empty strings for username and password when establishing the server connection with PvConnectServer().

Note: The connection established by passing empty strings for username and password is an insecure connection, and will not have sufficient rights to perform most of the other operations in DTI.

See Also

PvConnectServer()
PvCountDSNs()
PvCreateDSN2()
PvDeleteDSN()
PvDisconnect()
PvGetDSNEx()
PvListDSNs()
PvModifyDSN2()
PvStart()
PvStop()

PvGetEngineInformation()

Retrieves the information about the database engine for a given *hConnection*.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetEngineInformation(
   BTI_LONG            hConnection,
   BTI_CHAR_PTR        pserverClient,
   BTI_ULONG_PTR       pdbuApiVer,
   BTI_ULONG_PTR       pmajor,
   BTI_ULONG_PTR       pminor,
   BTI_ULONG_PTR       pserverClientType);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pserverClient*	Address of a BTI_CHAR_PTR True - MKDE_SERVR_ENGINE_CID False - MKDE_CLNT_ENGINE_CID
Out	*pdbuApiVer*	Version of the structures. Can be NULL
Out	*pmajor*	Major version - can be NULL.
Out	*pminor*	Minor version - can be NULL.
Out	*pserverClientType*	Only for MKDE_SRVR_ENGINE_CID. Returns one of the following: UNKNOWN_ENGINE_CLIENT (0) NT_SERVER (1) WIN32_CLIENT (3) UNIX_SERVER (4) CLIENT_CACHE (5) VXWIN_SERVER(6) VXLINUX_SERVER(7) REPORT_ENGINE(9)

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvDisconnect()
PvStop()

PvGetError()

Returns an error description string, describing the preceding error. This function is only for errors encountered in catalog functions.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvGetError(
   LPSTR           errorDesc,
   WORD*           size);
```

Arguments

In/Out	*errorDesc*	String that will contain the error description.
In/Out	*size*	Size of *errorDesc*. If the size is not large enough to contain the error description, an error is returned and the required size is contained in *size*.

Return Values

PCM_Success	The operation was successful.
PCM_errStringTooShort	The *size* parameter was not large enough to contain the error description. The required length is returned in the *size* argument.

Remarks

The *errorDesc* string is allocated by the caller.

The maximum size of the error description is specified in the constant `ERROR_LEN` found in the header file ddf.h.

See Also

PvStart()
PvStop()

PvGetFileHandlesData()

Retrieves all the file handle information related to an open file.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetFileHandlesData(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      fileName,
   BTI_ULONG_PTR     pCount);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*fileName*	Full path name of the file to be queried.
Out	*pCount*	Address of an unsigned long to receive the number of handles for the open file.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FILE_NOT_OPEN	Specified file is not currently open.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The information will be cached by DTI for subsequent calls related to file handles. This function would be called first for an open file before calling any other functions to get file handle

information. The cached information for the file handles will be freed when PvFreeOpenFilesData() is called.

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetOpenFilesData()
- Caller already has a valid open file name.

See Also

PvStart()
PvConnectServer()
PvGetOpenFilesData()
PvFreeOpenFilesData()
PvDisconnect()
PvStop()

PvGetFileHandleInfo()

Query the information for a file handle associated with an open file.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetFileHandleInfo(
   BTI_LONG            hConnection,
  BTI_CHAR_PTR        fileName,
   BTI_ULONG           sequence,
   PVFILEHDLINFO*      pFileHdlInfo);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*fileName*	Full path name of the file to be queried.
In	*sequence*	The sequence number (zero-based) of the file handle. Must be within a valid range with upper limit defined by the number of file handles obtained by PvGetFileHandlesData().
Out	*pFileHdlInfo*	Address of a PVFILEHDLINFO structure to receive the information on the file handle.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer
P_E_INVALID_SEQUENCE	Sequence number is not valid
P_E_FILE_NOT_OPEN	Specified file is not currently open.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetOpenFilesData()
- Data for open file handles retrieved by calling PvGetFileHandlesData();
- Caller already has a valid open file name.
- Caller already has a valid file handle sequence.

See Also

PvStart()
PvConnectServer()
PvGetOpenFilesData()
PvGetFileHandlesData()
PvGetOpenFileName()
PvFreeOpenFilesData()
PvDisconnect()
PvStop()

PvGetFileInfo()

Query the information for an open file.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetFileInfo(
   BTI_LONG            hConnection,
   BTI_CHAR_PTR        fileName,
   PVFILEINFO*         pFileInfo);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*fileName*	Full path name of the file to be queried.
Out	*pFileInfo*	Address of a PVFILEINFO structure to receive the information on the file.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer
P_E_FILE_NOT_OPEN	Specified file is not currently open.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetOpenFilesData();
- Caller already has a valid open file name.

See Also

PvStart()
PvConnectServer()
PvGetOpenFilesData()
PvDisconnect()
PvStop()

PvGetLongValue()

Retrieves the value for a long integer type setting, from the data source specified by *whichData*.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetLongValue(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_LONG_PTR      pValue,
   BTI_SINT          whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
Out	*pValue*	Address of a long integer variable that receives the setting value.
In	*whichData*	Flag to indicate which value is requested: PVDATA_DEFAULT returns default value. PVDATA_CURRENT returns current value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_INVALID_DATA_TYPE	The requested setting is not of long integer type.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

To obtain the minimum and maximum values that the setting can accept, use the PvGetValueLimit() function.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetValueLimit()
PvDisconnect()
PvStop()

PvGetMkdeClientId()

Get the client ID of an active MicroKernel Engine client.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeClientId(
   BTI_LONG            hConnection,
   BTI_ULONG           sequence,
   PVCLIENTID*         pClientId);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*sequence*	The sequence number (zero based) of the MicroKernel Engine client. Must be within a valid range with upper limit returned by PvGetMkdeClientsData().
Out	*pClientId*	Address of the PVCLIENTID structure to hold the returned client ID information.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_SEQUENCE	Sequence number is not valid.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for active clients retrieved by calling PvGetMkdeClientsData()

See Also

PvStart()
PvConnectServer()
PvGetMkdeClientsData()
PvGetMkdeClientInfo()
PvFreeMkdeClientsData()
PvDisconnect()
PvStop()

PvGetMkdeClientInfo()

Query the information for an active MicroKernel Engine client.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeClientInfo(
   BTI_LONG            hConnection,
   PVCLIENTID*         pClientId,
   PVMKDECLIENTINFO*   pClientInfo);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pClientId*	Address of the PVCLIENTID structure to identify the MicroKernel Engine client.
Out	*PClientInfo*	Address of a PVMKDECLIENTINFO structure to receive the information for the MicroKernel Engine client.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_CLIENT	Invalid client ID.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for active clients retrieved by calling PvGetMkdeClientsData().
- Caller already has a valid active MicroKernel Engine client ID.

See Also

PvStart()
PvConnectServer()
PvGetMkdeClientsData()
PvGetMkdeClientId()
PvFreeMkdeClientsData()
PvDisconnect()
PvStop()

PvGetMkdeClientHandlesData()

Retrieves the number of MicroKernel Engine client handles related to an active client.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeClientHandlesData(
   BTI_LONG            hConnection,
   PVCLIENTID*         pClientId,
   BTI_ULONG_PTR       pCount);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pClientId*	Address of the PVCLIENTID structure to identify the MicroKernel Engine client.
Out	*pCount*	Address of an unsigned long to receive the number of handles for the MicroKernel Engine client.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to MicroKernel Engine clients not available.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

When you call this function, all information regarding MicroKernel Engine client handles is cached by DTI for subsequent function calls related to client handles. If you want to obtain other information about clients, see PvGetMkdeClientsData().

This function should be called first before calling any other functions that return client handle information.

The cached information for the MicroKernel Engine client handles will be freed along with the information about the clients when PvFreeMkdeClientsData() is called.

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for active clients retrieved by calling PvGetMkdeClientsData().
- Caller already has a valid active MicroKernel Engine client ID.

See Also

PvStart()
PvConnectServer()
PvGetMkdeClientsData()
PvFreeMkdeClientsData()
PvDisconnect()
PvStop()

PvGetMkdeClientHandleInfo()

Query the information for a MicroKernel Engine client handle associated with an active client.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeClientHandleInfo(
   BTI_LONG               hConnection,
   PVCLIENTID*            pClientId,
   BTI_ULONG              sequence,
   PVMKDECLIENTHDLINFO*   pClientHdlInfo);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pClientId*	Address of the PVCLIENTID structure to identify the MicroKernel Engine client.
In	*sequence*	The sequence number (zero based) of the client handle. Must be within a valid range with upper limit defined by the number of handles obtained by PvGetMkdeClientHandlesData().
Out	*pClientHdlInfo*	Address of a PVMKDECLIENTHDLINFO structure to receive the information on the client handle.

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_INVALID_CLIENT	Invalid client ID.
P_E_INVALID SEQUENCE	Sequence number is not valid.
P_E_FAIL	Failed to disconnect to the named server.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for active MicroKernel Engine clients retrieved by calling PvGetMkdeClientsData();
- Data for MicroKernel Engine client handles retrieved by calling PvGetMkdeClientHandlesData();
- Caller already has a valid active MicroKernel Engine client ID.
- Caller already has a valid handle sequence for the active MicroKernel Engine client.

See Also

PvStart()
PvConnectServer()
PvGetMkdeClientsData()
PvGetMkdeClientHandlesData()
PvFreeMkdeClientsData()
PvDisconnect()
PvStop()

PvGetMkdeClientsData()

Retrieves all the information related to the active MicroKernel Engine clients.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeClientsData(
   BTI_LONG            hConnection,
   BTI_ULONG_PTR       pCount);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pCount*	Address of an unsigned long to receive the number of active MicroKernel Engine clients.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

When you call this function, all information regarding MicroKernel Engine clients is cached by DTI for subsequent function calls related to clients. The one exception is information regarding client handles, which is cached using a similar function PvGetMkdeClientHandlesData().

This function should be called first before calling any other functions that return client information. The caller should call PvFreeMkdeClientsData() to free the cached information when it is no longer needed.

This function can also be called to refresh the cached information.

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvFreeMkdeClientsData()
PvGetMkdeClientHandlesData()
PvDisconnect()
PvStop()

PvGetMkdeCommStat()

Retrieves all the MicroKernel Engine communication statistics data.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeCommStat(
   BTI_LONG            hConnection,
   PVCOMMSTAT*         pCommStat);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pCommStat*	Address of a PVCOMMSTAT structure to receive the MicroKernel Engine communication statistics.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_COMPONENT_NOT_LOADED	Component is not loaded
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetSQLConnectionsData()

See Also

PvStart()
PvConnectServer()
PvGetSQLConnectionsData()
PvGetMkdeUsage()
PvFreeSQLConnectionsData()
PvDisconnect()
PvStop()

PvGetMkdeCommStatEx()

Retrieves all the MicroKernel Engine communication statistics data.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeCommStatEx(
   BTI_LONG              hConnection,
   PVCOMMSTATEX*         pCommStatEx);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pCommStatEx*	Address of a PVCOMMSTATEX structure to receive the MicroKernel Engine communication statistics.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_COMPONENT_NOT_LOADED	Component is not loaded
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

This function returns the same data as PvGetMkdeCommStat but uses a new structure PVCOMMSTATEX that contains two additional elements. The added elements (totalTimeouts and totalRecoveries) are related to the auto reconnect feature. See *Advanced Operations Guide* for more information on auto reconnect.

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetSQLConnectionsData()

See Also

PvStart()
PvConnectServer()
PvGetSQLConnectionsData()
PvGetMkdeUsage()
PvFreeSQLConnectionsData()
PvDisconnect()
PvStop()

PvGetMkdeUsage()

Retrieves the resource usage information from the MicroKernel Engine, including current, peak, and maximum settings for licenses, files, handles, transactions, clients, threads, and locks.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeUsage(
   BTI_LONG            hConnection,
   PVMKDEUSAGE*        pMkdeUsage);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pMkdeUsage*	Address of a PVMKDEUSAGE structure to receive the MicroKernel Engine resource usage information.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetMkdeCommStat()
PvGetMkdeUsageEx()
PvDisconnect()
PvStop()

PvGetMkdeUsageEx()

Retrieves the resource usage information from the MicroKernel Engine database engine, including current, peak, and maximum settings for use count, session count, data in use, files, handles, transactions, clients, threads, and locks, and the duration, in seconds, that the database engine has been running (referred to as "engine uptime").

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeUsageEx(
   BTI_LONG             hConnection,
   PVMKDEUSAGEEX*       pMkdeUsage);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pMkdeUsage*	Address of a PVMKDEUSAGEEX structure to receive the MicroKernel Engine resource usage information.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

This function, PvGetMkdeUsageEx(), is similar to PvGetMkdeUsage(); the only difference is in the structures. While supplying the same elements, PVMKDEUSAGEEX supplies four-byte elements when PVMKDEUSAGE supplies two-byte ones.

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetMkdeCommStat()
PvGetMkdeUsage()
PvDisconnect()
PvStop()

PvGetMkdeVersion()

Retrieves the MicroKernel Engine version information.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetMkdeVersion(
   BTI_LONG            hConnection,
   PVVERSION*          pMkdeVersion);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pMkdeVersion*	Address of a PVVERSION structure to receive the MicroKernel Engine version information.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_COMPONENT_NOT_LOADED	Component not loaded.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetMkdeCommStat()
PvGetMkdeUsageEx()
PvDisconnect()
PvStop()

PvGetOpenFilesData()

Retrieves all the information related to the open files.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetOpenFilesData(
   BTI_LONG            hConnection,
   BTI_ULONG_PTR       pCount);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pCount*	Address of an unsigned long to receive the number of open files.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

The information will be cached by DTI for subsequent calls related to open files. This function should be called first before calling any other functions to get open file information.

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

The following post condition must be met:

- The caller should call PvFreeOpenFilesData() to free the cached information when it is no longer needed.

See Also

PvStart()
PvConnectServer()
PvGetOpenFileName()
PvFreeOpenFilesData()
PvDisconnect()
PvStop()

PvGetOpenFileName()

Retrieves the full path name of an open file.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetOpenFileName(
   BTI_LONG          hConnection,
   BTI_ULONG         sequence,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      fileName);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*sequence*	The sequence number (zero based) of the file. Must be within a valid range with upper limit returned by PvGetOpenFilesData().
In/Out	*pBufSize*	Address of an unsigned long containing size of buffer allocated to receive the file name. Receives actual size of chars copied. The size should include the null terminator.
In/Out	*fileName*	String value returned.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_DATA_UNAVAILABLE	Data related to active clients not available.
P_E_NULL_PTR	Call with NULL pointer.
P_E_BUFFER_TOO_SMALL	Allocated buffer is too small for the string, returned string is truncated. In this case the required size is in pBufSize.
P_E_INVALID_SEQUENCE	Sequence number is not valid.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for open files retrieved by calling PvGetOpenFilesData().

See Also

PvStart()
PvConnectServer()
PvGetOpenFilesData()
PvFreeOpenFilesData()
PvDisconnect()
PvStop()

PvGetProductsInfo()

Retrieves xml string with information on all Zen products found by the License Manager.

Header File: dtilicense.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvGetProductsInfo (
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      productInfo,
   BTI_ULONG_PTR     pBufSize);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*productInfo*	XML string returned with product information.
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer allocated to receive the string. It receives the actual length of selection string.

Return Values

DBU_SUCCESS	The operation was successful.
P_E_FAIL	Failed for other reasons.
Status code pertaining to license administration or to authorization	See *Status Codes and Messages* for License Administrator Status Codes and Authorization Status Codes.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Product Information Returned by PvGetProductsInfo()

Following is the document type definition (DTD) for the XML string returned by `PvGetProductsInfo()` and an explanation of its terms:

```
<!DOCTYPE products [
<!ELEMENT products (product*)>
<!ELEMENT product (name,id,licenses)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT licenses (license*)>
<!ELEMENT license
(type,productCode*,productKey*,state*,feature*,edition*,maxUserCount*,maxSessionCount*,maxDataInUseG
B*,platform*,sequence*,userCount*,sessionCount*,dataInUseGB*,timeStamp*,oemId*,application*,descript
ion*,isremovable*,gracePeriodEnd*)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT productCode (#PCDATA)>
<!ELEMENT productKey (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT feature (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT maxUserCount (#PCDATA)>
<!ELEMENT maxSessionCount (#PCDATA)>
<!ELEMENT maxDataInUseGB (#PCDATA)>
<!ELEMENT platform (#PCDATA)>
<!ELEMENT sequence (#PCDATA)>
<!ELEMENT userCount (#PCDATA)>
<!ELEMENT sessionCount (#PCDATA)>
<!ELEMENT dataInUseGB (#PCDATA)>
<!ELEMENT timeStamp (#PCDATA)>
<!ELEMENT oemId (#PCDATA)>
<!ELEMENT application (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT isremovable (#PCDATA)>
<!ELEMENT gracePeriodEnd (#PCDATA)>
]>
```

products	A container for all products returned by PvGetProductsInfo().
product	A container for information about a single product.
name	The name of the product.
id	The Zen code for the product. Refer to the dtilicense header file for the list of product codes returned.
licenses	A container for all licenses that apply to the product.
license	A container for information about a single license.

type	The license type: 1: Permanent 2: Expiring license set at issue date 4: Expiring license applied at install time 7: User count increase
productCode	The Zen code for the product. See the dtilicense header file for the list of product codes returned.
productKey	The key used for product authorization. Can be empty if product authorization was not used.
state	The current state of the license: 0: Active 1: Expired 2: Disabled 3: Inactive 4: Failed validation
feature	Reserved.
edition	Reserved.
maxUserCount	Maximum concurrent users allowed. Zero indicates unlimited users on Zen Enterprise Server and Workgroup editions. Not applicable on Zen Cloud Server edition and always returns zero.
maxSessionCount	Maximum concurrent sessions allowed. Zero indicates unlimited sessions on the Zen Cloud Server edition. Not applicable on Zen Enterprise Server and Workgroup editions and always returns zero.
maxDatainUseGB	Maximum amount of data allowed to be used simultaneously, measured in gigabytes. Zero indicates unlimited amount of data on Zen Cloud Server edition. Not applicable on Zen Enterprise Server and Workgroup editions and always returns zero.

platform	The supported platforms: 0: ANY 1: WIN 2: WIN32 3: WIN64 4: LINUX 5: LINUX32 6: LINUX64 7: MAC 8: MAC32 9: MAC64
sequence	The license sequence number.
userCount	The number of users permitted by the license. -1 indicates unlimited number of users on Zen Enterprise Server and Workgroup editions. Not applicable on Zen Cloud Server edition and always returns zero.
sessionCount	The number of sessions permitted by the license. -1 indicates unlimited number of users on Zen Cloud Server editions. Not applicable on Zen Enterprise Server and Workgroup editions and always returns zero.
dataInUseGB	The amount of data in use permitted by the license, measured in gigabytes. -1 indicates unlimited data count size on Zen Cloud Server editions. Not applicable on Zen Enterprise Server and Workgroup editions and always returns zero.
timeStamp	For temporary keys, the expiration day represented as the number of days from January 1, 2000.
oemId	The vendor ID.
application	The vendor's application ID.
description	Reserved.
isremovable	The license key is removable: 0: Not removable 1: Removable
gracePeriodEnd	Number of days remaining before the engine is disabled for failing license validation. Empty if a failed-validation period is not applicable to this product. -1 if a failed-validation period is applicable but not in effect for this product.

Example

```
<?xml version="1.0" encoding='UCS-4' ?>
<!DOCTYPE products [
<!ELEMENT products (product*)>
<!ELEMENT product (name,id,licenses)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT licenses (license*)>
<!ELEMENT license
(type,productCode*,productKey*,state*,feature*,edition*,maxUserCount*,maxSessionCount*,maxDataInUseG
B*,platform*,sequence*,userCount*,sessionCount*,dataInUseGB*,timeStamp*,oemId*,application*,descript
ion*,isremovable*,gracePeriodEnd*)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT productCode (#PCDATA)>
<!ELEMENT productKey (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT feature (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT maxUserCount (#PCDATA)>
<!ELEMENT maxSessionCount (#PCDATA)>
<!ELEMENT maxDataInUseGB (#PCDATA)>
<!ELEMENT platform (#PCDATA)>
<!ELEMENT sequence (#PCDATA)>
<!ELEMENT userCount (#PCDATA)>
<!ELEMENT sessionCount (#PCDATA)>
<!ELEMENT dataInUseGB (#PCDATA)>
<!ELEMENT timeStamp (#PCDATA)>
<!ELEMENT oemId (#PCDATA)>
<!ELEMENT application (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT isremovable (#PCDATA)>
<!ELEMENT gracePeriodEnd (#PCDATA)>
]>
<products>
  <product>
    <name>DataExchange 5 Server: Real-Time Backup</name>
    <id>78</id>
    <licenses>
      <license>
        <type>1</type>
        <productCode>78</productCode>
        <productKey> ABCDE-55555-FGHIJ-55555-KLMNO-55555</productKey>
        <state>0</state>
        <feature>0</feature>
        <edition>0</edition>
        <maxUserCount>0</maxUserCount>
        <maxSessionCount>0</maxSessionCount>
        <maxDataInUseGB>0</maxDataInUseGB>
        <platform>2</platform>
        <sequence>0</sequence>
        <userCount>1</userCount>
        <sessionCount>0</sessionCount>
        <dataInUseGB>0</dataInUseGB>
        <timeStamp>0</timeStamp>
        <oemId>0</oemId>
        <application>0</application>
        <description></description>
        <isremovable>1</isremovable>
        <gracePeriodEnd>-1</gracePeriodEnd>
      </license>
    </licenses>
  </product>
  <product>
```

```
    <name>PSQL 12 Server</name>
    <id>425</id>
    <licenses>
      <license>
        <type>2</type>
        <productCode>425</productCode>
        <productKey></productKey>
        <state>0</state>
        <feature>0</feature>
        <edition>0</edition>
        <maxUserCount>0</maxUserCount>
        <maxSessionCount>0</maxSessionCount>
        <maxDataInUseGB>0</maxDataInUseGB>
        <platform>2</platform>
        <sequence>0</sequence>
        <userCount>10</userCount>
        <sessionCount>0</sessionCount>
        <dataInUseGB>0</dataInUseGB>
        <timeStamp>4489</timeStamp>
        <oemId>8</oemId>
        <application>604</application>
        <description></description>
        <isremovable>0</isremovable>
        <gracePeriodEnd></gracePeriodEnd>
      </license>
      <license>
        <type>4</type>
        <productCode>425</productCode>
        <productKey></productKey>
        <state>0</state>
        <feature>0</feature>
        <edition>0</edition>
        <maxUserCount>0</maxUserCount>
        <maxSessionCount>0</maxSessionCount>
        <maxDataInUseGB>0</maxDataInUseGB>
        <platform>1</platform>
        <sequence>11200</sequence>
        <userCount>20</userCount>
        <sessionCount>0</sessionCount>
        <dataInUseGB>0</dataInUseGB>
        <timeStamp>4429</timeStamp>
        <oemId>0</oemId>
        <application>1</application>
       <description></description>
        <isremovable>0</isremovable>
        <gracePeriodEnd></gracePeriodEnd>
      </license>
      <license>
        <type>1</type>
        <productCode>425</productCode>
        <productKey>ABCDE-55555-FGHIJ-55555-KLMNO-55555</productKey>
        <state>0</state>
        <feature>0</feature>
        <edition>0</edition>
        <maxUserCount>0</maxUserCount>
        <maxSessionCount>0</maxSessionCount>
        <maxDataInUseGB>0</maxDataInUseGB>
        <platform>2</platform>
        <sequence>0</sequence>
        <userCount>10</userCount>
        <sessionCount>0</sessionCount>
        <dataInUseGB>0</dataInUseGB>
        <timeStamp>0</timeStamp>
        <oemId>333</oemId>
       <application>334</application>
```

```
        <description></description>
        <isremovable>1</isremovable>
        <gracePeriodEnd>-1</gracePeriodEnd>
      </license>
    </licenses>
  </product>
</products>
```

See Also

PvValidateLicenses()
PvConnectServer()
PvStart()
PvStop()

PvGetSelectionString()

Retrieves display string for a specific choice of selection type setting.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSelectionString(
    BTI_LONG            hConnection,
    BTI_ULONG           settingID,
    BTI_ULONG           selection,
    BTI_ULONG_PTR       pBufSize,
    BTI_CHAR_PTR        dispString);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In	*selection*	Selection choice index. PSelectionList returned from PvGetAllPossibleSelections().
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer allocated to receive the string. It receives the actual length of selection string.
Out	*dispString*	Display string returned.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The requested setting is not of selection type.
P_E_BUFFER_TOO_SMALL	The array size is too small. In this case, the required size is returned in *pBufSize*.

P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetSelectionStringSize()
PvDisconnect()
PvStop()

PvGetSelectionStringSize()

Retrieves size of buffer needed for successful PvGetSelectionString () call.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSelectionStringSize(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pBufSize);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer in PvGetSelectionString() call allocated to receive the string. It receives the actual length of selection string.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The requested setting is not of selection type.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvDisconnect()
PvStop()

PvGetSelectionValue()

Retrieves the value for a selection type setting, from the data source specified by *whichData*.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSelectionValue(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pNumItems,
   BTI_LONG_PTR      pValue,
   BTI_SINT          whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pNumItems*	Address of an unsigned long that specifies the array size on input, and receives the number of individual selection items on return.
Out	*pValue*	Array of individual selection indexes.
In	*whichData*	Flag to indicate which value is requested: PVDATA_DEFAULT returns default value. PVDATA_CURRENT returns current value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The requested setting is not of string type.

P_E_BUFFER_TOO_SMALL	The array size is too small. In this case, the required size is returned in *pNumItems*.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvDisconnect()
PvStop()

PvGetServerName()

Retrieves the name of the connected server indicated by the connection handle.

Header File: connect.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetServerName(
   BTI_LONG          hConnection,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      serverName);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer allocated to receive server name.
In/Out	*serverName*	Returned server name if successful, empty string otherwise.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in PBufSize.
P_E_FAIL	Failed to connect to the named server.

Remarks

The implementation should perform the necessary initializations when called the first time.

Multiple simultaneous connections are allowed.

See Also

PvStart()
PvConnectServer()
PvDisconnect()
PvStop()

PvGetSettingHelp()

Retrieves help string related to setting.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingHelp(
   BTI_ULONG          settingID,
   BTI_ULONG_PTR      pBufSize,
   BTI_CHAR_PTR       pHelpString);
```

Arguments

In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pBufSize*	Address of an unsigned long containing size of buffer allocated to receive setting value. Receives actual size of setting value. The size should include the NULL terminator.
Out	*pHelpString*	String value returned.

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.
P_E_BUFFER_TOO_SMALL	The buffer allocated is too small and the display string is truncated. In this case, the required buffer size is returned in *pBufSize*.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetSettingInfo()
PvDisconnect()
PvStop()

PvGetSettingHelpSize()

Retrieves help string related to setting.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingHelpSize(
   BTI_ULONG          settingID,
   BTI_ULONG_PTR      pBufSize);
```

Arguments

In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pBufSize*	Address of an unsigned long containing size of buffer allocated to receive setting value. Receives actual size of setting value. The size should include the NULL terminator.

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetSettingInfo()
PvDisconnect()
PvStop()

PvGetSettingInfo()

Retrieves setting information for a setting.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingInfo(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   PVSETTINGINFO*    pSettingInfo);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
Out	*pSettingInfo*	Address of a PVSETTINGINFO structure that receives setting information.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetSettingHelp()
PvDisconnect()
PvStop()

PvGetSettingList()

Retrieves a list of settings belonging to the specified category.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingList(
   BTI_LONG          hConnection,
   BTI_ULONG         categoryID,
   BTI_ULONG_PTR     pNumSettings,
   BTI_ULONG_PTR     pSettingList);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*categoryID*	Unique identifier for the category
Out	*pNumSettings*	Address of an unsigned long containing size of the array on input, and receives number of items in the returned list.
Out	*pSettingList*	Pointer to the list of returned setting IDs.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_BUFFER_TOO_SMALL	The array size is too small. In this case, the required size is returned in *pNumSettings*.
P_E_FAIL	Failed for other reasons.

Remarks

If the connection is a remote connection, only server-side settings for the category are returned. If the connection is a local connection, both client-side and server-side settings for this category will be returned.

Use PvIsSettingAvailable() to determine if the setting can be set at this time.

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvIsSettingAvailable()
PvGetSettingHelp()
PvGetSettingInfo()
PvGetSettingMap()
PvGetSettingUnits()
PvDisconnect()
PvStop()

PvGetSettingListCount()

Retrieves number of settings belonging to the specified category. This number can then be used to allocate an array to pass to PvGetSettingList().

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingListCount(
   BTI_LONG          hConnection,
   BTI_ULONG         categoryID,
   BTI_ULONG_PTR     pNumSettings);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*categoryID*	Unique identifier for the category.
Out	*pNumSettings*	Address of an unsigned long containing size of the array on input, and receives number of items in the returned list.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

If the connection is a remote connection, only server-side settings for the category are returned. If the connection is a local connection, both client-side and server-side settings for this category will be returned.

Use PvIsSettingAvailable() to determine if the setting can be set at this time.

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvIsSettingAvailable()
PvGetSettingHelp()
PvGetSettingInfo()
PvGetSettingMap()
PvGetSettingUnits()
PvDisconnect()
PvStop()

PvGetSettingMap()

Retrieves option ID and component ID for a setting.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingMap(
   BTI_ULONG          settingID,
   BTI_WORD_PTR       pComponentID,
   BTI_WORD_PTR       pOptionID);
```

Arguments

In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
Out	*pComponentID*	Address of an unsigned short for Component.
Out	*pOptionID*	Address of an unsigned short for Option

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

Option and Component maps setting to DBUGetInfo or DBUSetInfo calls.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()

PvGetSettingList()
PvDisconnect()
PvStop()

PvGetSettingUnits()

Retrieves default units and suggested factor. This function is only valid for settings of long integer type.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingUnits(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      pValue,
   BTI_ULONG_PTR     pFactor,
   BTI_ULONG_PTR     pFBufSize,
   BTI_CHAR_PTR      pFValue);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer allocated to receive string of default units. Receives actual size of string of default units. The size should include the NULL terminator.
Out	*pValue*	String of default value returned.
Out	*pFactor*	Address of an unsigned long for factor.
In/Out	*pFBufSize*	Address of an unsigned long containing size of buffer allocated to receive string of "factor" units. Receives actual size of string of default units. The size should include the NULL terminator.
Out	*pFValue*	String of "factor" value returned.

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The setting requested is not of long integer type.
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in *pBufSize*.
P_E_FAIL	Failed to connect to the named server.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvDisconnect()
PvStop()

PvGetSettingUnitsSize()

Returns the size in bytes of buffer size required to receive information in PvGetSettingUnits() call.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSettingUnitsSize(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pBufSize,
   BTI_ULONG_PTR     pFBufSize);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pBufSize*	Address of an unsigned long containing size of the buffer allocated to receive string of default units. Receives actual size of string of default units. The size should include the NULL terminator.
In/Out	*pFBufSize*	Address of an unsigned long containing size of buffer allocated to receive string of "factor" units. Receives actual size of string of default units. The size should include the NULL terminator.

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The setting requested is not of long integer type.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvDisconnect()
PvStop()

PvGetSQLConnectionsData()

Retrieves the number of connections to the SQL Connection Manager and all information related to the connections.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSQLConnectionsData(
   BTI_LONG          hConnection,
   BTI_ULONG_PTR     pCount);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
Out	*pCount*	Address of an unsigned long to receive the number of SQL connections.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_FAIL	Failed for other reasons.

Remarks

The information will be cached by DTI for subsequent calls related to SQL connections. This function should be called first before calling any other functions to get SQL connection information.

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

The following post conditions must be met:

- The caller should call PvFreeSQLConnectionsData() to free the cached information when it is no longer needed.

See Also

PvStart()
PvConnectServer()
PvGetMkdeCommStat()
PvGetSQLConnectionInfo()
PvFreeSQLConnectionsData()
PvDisconnect()
PvStop()

PvGetSQLConnectionInfo()

Query the information for a SQL connection.

Header File: monitor.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetSQLConnectionInfo(
   BTI_LONG            hConnection,
   BTI_ULONG           sequence,
   PVSQLCONNINFO*      pSQLConnInfo);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*sequence*	The sequence number (zero based) of the SQL connection. Must be within a valid range with upper limit defined by the number of SQL connections obtained by PvGetSQLConnectionsData().
Out	*pSQLConnInfo*	Address of a PVSQLCONNINFO structure to receive the information on the SQL connection.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	*hConnection* parameter is not a valid connection handle.
P_E_DATA_UNAVAILABLE	Data not available for the SQL connection.
P_E_NULL_PTR	*pSQLConnInfo* pointer is NULL.
P_E_INVALID_SEQUENCE	Sequence number is not valid.
P_E_FAIL	Failed to disconnect to the named server.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Data for SQL connections retrieved by calling PvGetSQLConnectionsData()
- Caller already has a valid SQL connection sequence.

See Also

PvStart()
PvConnectServer()
PvGetSQLConnectionsData()
PvFreeSQLConnectionsData()
PvDisconnect()
PvStop()

PvGetStringType()

Retrieves additional information about PVSETTING_STRING setting which only applies to string type setting.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetStringType(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pTypeString);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
Out	*pTypeString*	Subtype of PVSETTING_STRING returned.

Return Values

P_OK	The operation was successful.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The setting requested is not of string type.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Here are possible subtypes of PVSETTING_STRING:

- PVSTRING – a string that is neither dir or file
- PVFILESTRING – a string that indicates the path to a file
- PVDIRECTORYSTRING – a string that indicates a directory

The subtypes are defined in config.h.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetStringValue()
PvSetStringValue()
PvDisconnect()
PvStop()

PvGetStringValue()

Retrieves the value (Null terminated string) for a string type setting, from the data source specified by *whichData*. Some settings may return a list of strings separated by semicolons (;).

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetStringValue(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pBufSize,
   BTI_CHAR_PTR      value,
   BTI_SINT          whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pBufSize*	Address of an unsigned long containing the size of the buffer allocated to receive the setting value. Receives the actual size of setting value.
Out	*value*	Address of a long integer variable that receives the setting value.
In	*whichData*	Flag to indicate which value is requested: PVDATA_DEFAULT returns default value. PVDATA_CURRENT returns current value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The requested setting is not of string type.

P_E_BUFFER_TOO_SMALL	Allocated buffer is too small for the string (the return string is truncated). In this case, the required size is returned in *pBufSize*.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetStringType()
PvSetStringValue()
PvGetStringValueSize()
PvDisconnect()
PvStop()

PvGetStringValueSize()

Retrieves the value (Null terminated string) for a string type setting, from the data source specified by *whichData*. Some settings may return a list of strings separated by semicolons (;).

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetStringValueSize(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG_PTR     pBufSize,
   BTI_SINT          whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In/Out	*pBufSize*	Address of an unsigned long containing the size of the buffer allocated to receive the setting value. Receives the actual size of setting value.
In	*whichData*	Flag to indicate which value is requested: PVDATA_DEFAULT returns default value. PVDATA_CURRENT returns current value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The requested setting is not of string type.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetStringType()
PvSetStringValue()
PvDisconnect()
PvStop()

PvGetTable()

Returns table attributes for a given table.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvGetTable(
   WORD             dictHandle,
   LPSTR            tableName,
   TABLEINFO**      tableProps,
   COLUMNMAP**      columnList,
   WORD*            columnCount,
   INDEXMAP**       indexList,
 WORD*            indexCount);
```

Arguments

In	*dictHandle*	Handle of an open dictionary returned by PvOpenDatabase().
In	*tableName*	Name of table to retrieve.
Out	*tableProps*	Structure containing table information.
Out	*columnList*	Array of columns defined in the table.
Out	*columnCount*	Number of columns in columnList.
Out	*indexList*	Array of segments defined in the table.
Out	*indexCount*	Number of indexes in the *indexList* array.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	A general failure occurred
PCM_errMemoryAllocation	Error during memory allocation
PCM_errInvalidDictionaryHandle	The specified dictionary handle does not exist.

Remarks

You must first open a dictionary successfully using PvOpenDatabase().

TableProps, indexList, and columnList arrays will need to be released using PvFreeTable.

See Also

PvStart()
PvOpenDatabase()
PvGetTableNames()
PvFreeTable()
PvFreeTableNames()
PvCloseDictionary()
PvStop()

PvGetTableNames()

Returns table names of all the tables in the open data dictionary.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvGetTableNames(
   WORD             dictHandle,
   TABLEMAP**       tableList,
   WORD*            tableCount);
```

Arguments

In	*dictHandle*	Handle of an open dictionary returned by PvOpenDatabase().
Out	*tableList*	Array of TABLEMAP structures that contain table names.
Out	*tableCount*	Number of table names returned in *tableList*.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errInvalidDictionaryHandle	The specified dictionary handle obtained by PvOpenDatabase() is invalid.

Remarks

You must first open a dictionary successfully using PvOpenDatabase().

TableList array will need to be released using PvFreeTableNames().

You can retrieve more information about a specific table using PvGetTable().

See Also

PvStart()
PvOpenDatabase()
PvGetTable()
PvFreeTable()
PvFreeTableNames()
PvCloseDictionary()
PvStop()

PvGetTableStat()

Returns statistical information on a given table.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvGetTableStat(
   BTI_WORD               dbHandle,
   const BTI_CHAR*        tableName,
   TABLESTAT*             tableStat);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*tableName*	Table name for which you want statistical information.
Out	*tableStat*	TABLESTAT structure containing table statistics information.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errInvalidDictionaryHandle	The specified dictionary handle obtained by PvOpenDatabase() is invalid.
PCM_errTableNotFound	The specified table was not found.

Remarks

You must first obtain a database handle using PvOpenDatabase().

You can retrieve more information about a specific table using PvGetTable().

If the number of records in the data file is greater than the maximum value that the TABLESTAT structure can return, then the maximum possible value is returned instead, which is 65535 as a 2-byte unsigned integer.

See Also

PvCloseDatabase()
PvFreeTable()
PvFreeTableNames()
PvGetTable()
PvGetTableStat2()
PvGetTableStat3()
PvOpenDatabase()
PvStart()
PvStop()

PvGetTableStat2()

Returns statistical information on a given table, including whether its data file is using compressed data pages. See also Creating a File with Page Level Compression in *Zen Programmer's Guide* and Record and Page Compression in *Advanced Operations Guide*.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvGetTableStat2   (
   BTI_WORD               dbHandle,
   const BTI_CHAR*        tableName,
   TABLESTAT2*             tableStat2);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*tableName*	Table name for which you want statistical information.
Out	*tableStat*	TABLESTAT2 structure containing table statistics information.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errInvalidDictionaryHandle	The specified dictionary handle obtained by PvOpenDatabase() is invalid.
PCM_Success	The operation was successful.

Remarks

You must first obtain a database handle using PvOpenDatabase().

You can retrieve more information about a specific table using PvGetTable().

For more information see Differences Between TABLESTAT2 and TABLESTAT.

If the number of records in the data file is greater than the maximum value that the TABLESTAT2 structure can return, then the maximum possible value is returned instead, which is 2,147,483,647 as a 4-byte signed integer.

See Also

PvGetTable()
PvGetTableStat()
PvGetTableStat3()
PvStart()
PvOpenDatabase()
PvOpenDatabase()
PvGetTable()
PvFreeTable()
PvFreeTableNames()
PvCloseDictionary()
PvCloseDatabase()
PvStop()

PvGetTableStat3()

Returns statistical information on a given table, including a 64-bit record count capable of indicating up to 2^{63}–1 records, or 9223372036854775807.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvGetTableStat3   (
   BTI_WORD               dbHandle,
   const BTI_CHAR*        tableName,
   TABLESTAT3*             tableStat3);
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*tableName*	Table name for which you want statistical information.
Out	*tableStat*	TABLESTAT3 structure containing table statistics information.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errInvalidDictionaryHandle	The specified dictionary handle obtained by PvOpenDatabase() is invalid.
PCM_errTableNotFound	The specified table was not found

Remarks

You must first obtain a database handle using PvOpenDatabase().

You can retrieve more information about a specific table using PvGetTable().

For more information see Differences Between TABLESTAT3 and TABLESTAT2.

See Also

PvGetTable()
PvGetTableStat()
PvGetTableStat2()
PvStart()
PvOpenDatabase()
PvOpenDatabase()
PvGetTable()
PvFreeTable()
PvFreeTableNames()
PvCloseDictionary()
PvCloseDatabase()
PvStop()

PvGetValueLimit()

Retrieves upper and lower limits for settings of long type.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvGetValueLimit(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_LONG_PTR      pMaxValue,
   BTI_LONG_PTR      pMinValue);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
Out	*pMaxValue*	Address of a long integer that receives the upper limit value. If NULL is passed here, no value will be returned. If a negative value is returned, interpret it as follows: /* Maximum valid memory or disk size */P_MAX_MEM_DISK_SIZE -129 /* Maximum size limited by available disk space */ P_MAX_LIMITED_BY_DISK -2 /* Maximum size limited by available memory */ P_MAX_LIMITED_BY_MEMORY -1
Out	*pMinValue*	Address of a long integer that receives the lower limit value. If NULL is passed here, no value will be returned.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.

P_E_INVALID_DATA_TYPE	The requested setting is not of long type.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetLongValue()
PvSetLongValue()
PvDisconnect()
PvStop()

PvIsDatabaseSecured()

Determines whether a given database has security enabled.

Header File: dtisecurity.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvIsDatabaseSecured(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_LONG_PTR      dbAuthentication);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database to check.
Out	*dbAuthentication*	2 if database is secured using domain authentication 1 if database is secured using Zen database authentication 0 if database is not secured

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_FAIL	Failed to open the database for other reasons.

Remarks

The following preconditions must be met:

- DTI session started by calling **PvStart()**.

- Connection established by PvConnectServer(), or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvOpenDatabase()
PvSecureDatabase()
PvSecureDatabase2()
PvUnSecureDatabase()
PvCloseDatabase()
PvDisconnect()
PvStop()

PvIsSettingAvailable()

Query to see if a setting is available for configuring.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvIsSettingAvailable(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting.

Return Values

Zero	Setting is unavailable.
Non-zero	Setting is available.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Setting may be unavailable due to insufficient rights to access the setting or if no such setting ID exist.

See Also

PvStart()
PvConnectServer()

PvGetCategoryList()
PvGetSettingList()
PvDisconnect()
PvStop()

PvListDSNs()

Retrieves the list of system datasource names (DSN) of type Pervasive ODBC Engine Interface.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to work with client DSNs.

Syntax

```
BTI_API PvListDSNs(
   BTI_LONG          hConnection,
   BTI_ULONG_PTR     pdsnListSize,
   BTI_CHAR_PTR      pdsnList,
   BTI_CHAR          filtering);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In/Out	*pdsnListSize*	Address of an unsigned long containing the size of the buffer for the list of DSNs. Receives actual size of the returned DSN list.
Out	*pdsnList*	Contains the list of DSNs if successful.
In	*filtering*	Set to 1 if you want only Zen DSNs. Set to 0 if you want all DSNs.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_BUFFER_TOO_SMALL	The buffer is too small for the string. In this case, the required buffer size is returned in *pdsnListSize*.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

To retrieve the list of DSNs without having to prompt the user to login, pass empty strings for username and password when establishing the server connection with PvConnectServer().

Note: The connection established by passing empty strings for username and password is an insecure connection, and will not have sufficient rights to perform most of the other operations in DTI.

Example

```
BTI_WORD           res = 0;
BTI_ULONG          dsncount = 0;
BTI_ULONG          dsnListSize = 0;
BTI_CHAR           * dsnList;

// MAX_DSN_NAME_LENGTH is defined to be 32
// in catalog.h
res = PvCountDSNs (hConnection,
                   &dsnCount,
                   1);
dsnlistSize = dsnCount * (MAX_DSN_NAME_LENGTH+1);
dsnList = new char[dsnListSize];
res = PvListDSNs (hConnection,
                  &dsnListSize,
                  dsnList,
                  1);
```

See Also

PvStart()
PvConnectServer()
PvCountDSNs()
PvGetDSN()
PvDisconnect()
PvStop()

PvModifyDatabase()

Modify an existing database using the specified information for the new database name, dictionary and data paths and the database flag.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvModifyDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbNameExisting,
   BTI_CHAR_PTR      dbNameNew,
   BTI_CHAR_PTR      dictPath,
   BTI_CHAR_PTR      dataPath,
   BTI_ULONG         dbFlags);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbNameExisting*	Name of the existing database
In	*dbNameNew*	Name of the new database. Set this parameter to NULL if you want the database name to remain unchanged.
In	*dictPath*	Dictionary path.
In	*dataPath*	Data path. Set this value to NULL to use the default data path (that is, the same as the dictionary path) If you want to modify a database to include MicroKernel Engine data files located in multiple paths, specify this parameter as a semicolon (;) delimited list. For example: `C:\data\path1;C:\data\path2`

In	*dbFlags*	Database flags, which can be a combination of the P_DBFLAG_ constants. • P_DBFLAG_RI (enforce integrity constraints, including referential integrity and triggers) • P_DBFLAG_BOUND (stamps the database name on the dictionary files so only that database can use them) • P_DBFLAG_DBSEC_AUTHENTICATION (use database security authentication, Mixed security policy. See Btrieve Security Policy.) • P_DBFLAG_DBSEC_AUTHORIZATION (use database security authorization, Database security policy. See Btrieve Security Policy.) • P_DBFLAG_LONGMETADATA (use V2 metadata. See Metadata Version.)

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_ACCESS_RIGHT	Insufficient access right for the operation
P_E_NOT_EXIST	Named database does not exist on the server.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer(), or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Btrieve Security Policy

The following table indicates how to specify a security model in a new database, or to interpret the security model of an existing database. Using any other combination of flags for security will result in status code 7024.

This Flag Combination	Represents This Security Model
No flags	Classic
P_DBFLAG_DBSEC_AUTHENTICATION	Mixed
P_DBFLAG_DBSEC_AUTHENTICATION P_DBFLAG_DBSEC_AUTHORIZATION	Database

See Also

PvStart()
PvConnectServer()
PvCreateDatabase()
PvGetDbNamesData()
PvGetDbName()
PvGetDbFlags()
PvGetDbDataPath()
PvGetDbDictionaryPath()
PvGetDbServerName()
PvFreeDbNamesData()
PvDisconnect()
PvStop()

PvModifyDatabase2()

Modify an existing database using the specified information for the new database name, dictionary and data paths, database flag, and code page. This function is the same as PvModifyDatabase() except that the database code page is also specified.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvModifyDatabase2(
   BTI_LONG           hConnection,
   BTI_CHAR_PTR       dbNameExisting,
   BTI_CHAR_PTR       dbNameNew,
   BTI_CHAR_PTR       dictPath,
   BTI_CHAR_PTR       dataPath,
   BTI_ULONG          dbFlags,
   BTI_LONG          dbCodePage);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbNameExisting*	Name of the existing database
In	*dbNameNew*	Name of the new database. Set this parameter to NULL if you want the database name to remain unchanged.
In	*dictPath*	Dictionary path.
In	*dataPath*	Data path. Set to NULL to use the default data path (that is, the same as the dictionary path) If you want to modify a database to include MicroKernel Engine data files located in multiple paths, specify this parameter as a semicolon (;) delimited list. For example: `C:\data\path1;C:\data\path2`

In	*dbFlags*	Database flags, which can be a combination of the P_DBFLAG_ constants. • P_DBFLAG_RI (enforce integrity constraints, including referential integrity and triggers) • P_DBFLAG_BOUND (stamps the database name on the dictionary files so only that database can use them) • P_DBFLAG_DBSEC_AUTHENTICATION (use database security authentication, Mixed security policy. See Btrieve Security Policy.) • P_DBFLAG_DBSEC_AUTHORIZATION (use database security authorization, Database security policy. See Btrieve Security Policy.) • P_DBFLAG_LONGMETADATA (use V2 metadata. See Metadata Version.)
In	*dbCodePage*	For databases on Windows platforms, a number indicating the code page for database data and metadata strings. For databases on Linux and macOS distributions, one of the following to indicate the code page for database data and metadata strings: • P_DBCODEPAGE_UTF8 • P_DBCODEPAGE_EUCJP • P_DBCODEPAGE_ISO8859_1 For databases on Windows, Linux, and macOS, the value can also be a zero or P_DBCODEPAGE_NA. A zero indicates legacy behavior. That is, no code page is specified, defaulting to the operating system encoding on the server machine. See also the Code Page database property in *Zen User's Guide*. P_DBCODEPAGE_NA specifies to leave the code page as is (the database code page is not to be changed). **Note:** The database engine does **not** validate the encoding of the data and metadata that an application inserts into a database. The engine assumes that all data was entered using the encoding of the server or the client as explained under Database Code Page and Client Encoding in *Advanced Operations Guide*.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer
P_E_ACCESS_RIGHT	Insufficient access right for the operation

P_E_NOT_EXIST	Named database does not exist on the server.
P_E_FAIL	Failed for other reasons.

Remarks

The following precondition must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer(), or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Btrieve Security Policy

See Btrieve Security Policy.

See Also

PvConnectServer()
PvCreateDatabase2()
PvCreateDSN2()
PvDisconnect()
PvFreeDbNamesData()
PvGetDbCodePage()
PvGetDbDataPath()
PvGetDbDictionaryPath()
PvGetDbFlags()
PvGetDbName()
PvGetDbNamesData()
PvGetDbServerName()
PvGetDSNEx2()
PvModifyDSN2()
PvStart()
PvStop()

PvModifyDSN()

Modifies an existing data source name.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav78.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to work with client DSNs.

Syntax

```
BTI_API PvModifyDSN(
   BTI_LONG       hConnection,
   BTI_CHAR_PTR   pdsnName,
   BTI_CHAR_PTR   pdsnDesc,
   BTI_CHAR_PTR   pdsnDBQ,
   BTI_LONG       openMode);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pdsnName*	Name of the DSN to modify.
In	*pdsnDesc*	New description for the DSN.
In	*pdsnDBQ*	New Database name for the DSN.
In	*openMode*	New Open mode for the DSN, which is one of the following: NORMAL_MODE ACCELERATED_MODE, READONLY_MODE EXCLUSIVE_MODE See also DSN Open Mode in *ODBC Guide*.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.

P_E_NULL_PTR	Call with NULL pointer.
P_E_DSN_DOES_NOT_EXIST	The specified DSN name does not exist.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_INVALID_OPEN_MODE	The specified open mode is invalid.
P_E_FAIL	Failed to retrieve data path.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvStart()
PvConnectServer()
PvListDSNs()
PvCreateDSN()
PvGetDSN()
PvGetDSNEx()
PvDeleteDSN()
PvCountDSNs()
PvStop()

PvModifyDSN2()

Modifies an existing data source name. This function is the same as PvModifyDSN() except that the encoding option for data is also specified.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

This function is deprecated in Zen v11 and higher versions. Use the ODBC API to work with client DSNs.

Syntax

```
BTI_API PvModifyDSN(
   BTI_LONG       hConnection,
   BTI_CHAR_PTR   pdsnName,
   BTI_CHAR_PTR   pdsnDesc,
   BTI_CHAR_PTR   pdsnDBQ,
   BTI_LONG       openMode,
   BTI_LONG       translate);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*pdsnName*	Name of the DSN to modify.
In	*pdsnDesc*	New description for the DSN.
In	*pdsnDBQ*	New Database name for the DSN.
In	*openMode*	Open mode for the DSN, which is one of the following: • NORMAL_MODE • ACCELERATED_MODE • READONLY_MODE • EXCLUSIVE_MODE See also DSN Open Mode in *ODBC Guide*.
In	translate	Encoding option for data, which can be one of the following: • DSNFLAG_DEFAULT • DSNFLAG_OEMANSI • DSNFLAG_AUTO See also Encoding Translation in *ODBC Guide*. Note that DSNFLAG_DEFAULT corresponds to the "None" encoding option in ODBC Administrator.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_DSN_DOES_NOT_EXIST	The specified DSN name does not exist.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_INVALID_OPEN_MODE	The specified open mode is invalid.
P_E_INVALID_TRANSLATE_OPTION	The specified encoding translation option is invalid.
P_E_FAIL	Failed to retrieve data path.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

See Also

PvConnectServer()
PvCountDSNs()
PvCreateDatabase2()
PvCreateDSN2()
PvDeleteDSN()
PvGetDSN()
PvGetDSNEx2()
PvListDSNs()
PvStart()
PvStop()

PvOpenDatabase()

Opens a database by name and returns a handle that can be used to manipulate the database catalog.

Header File: catalog.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvOpenDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR_PTR      dbUser,
   BTI_CHAR_PTR      dbPassword,
   BTI_WORD_PTR      dbHandle);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*dbName*	Name of the database.
In	*dbUser*	Database user name if security is defined.
In	*dbPassword*	Database password if security is defined.
Out	*dbHandle*	Returned handle to the database.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_FAIL	Failed to open the database for other reasons.
PCM_errSessionSecurityError	Invalid user name or password.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart()
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- If the database has security enabled, you must specify a valid database user name and password. Security for the returned database handle is enforced based on the access rights defined for the database, and should match behavior seen in SQL or ODBC access methods.

See Also

PvStart()
PvConnectServer()
PvGetDbFlags()
PvModifyDatabase()
PvCloseDatabase()
PvDropDatabase()
PvDisconnect()
PvStop()

PvOpenDictionary()

Opens an existing dictionary. Given an absolute path of the dictionary or data source names, it returns a dictionary handle that will be used for any subsequent calls to any functions.

Note: This function is deprecated in Zen 9 and higher versions. See PvOpenDatabase() to replace this function in your application.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT PvOpenDictionary(
   LPTSTR          path,
   WORD*           dictHandle,
   LPSTR           user,
   LPSTR           password);
```

Arguments

In	*path*	Fully-qualified path to the dictionary files.
Out	*dictHandle*	Handle to be used in subsequent calls
In	*user*	User name needed to open the dictionary. This argument can be set to NULL.
In	*password*	Used in conjunction with user name to open the dictionary files. Can also be NULL.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errMemoryAllocation	An error occurred during memory allocation.
PCM_errDictionaryPathNotFound	The specified dictionary path is invalid.
PCM_errDictionaryAlreadyOpen	The specified dictionary files are currently open.
PCM_SessionSecurityError	Either the user name or password is invalid.

Remarks

This function should be called first when accessing DDFs via DTI.

Multiple dictionaries can be open at one time.

Use PvCloseDictionary() to free the resources.

See Also

PvStart()
PvCreateDictionary()
PvCreateDatabase()
PvCloseDictionary()
PvStop()

PvRemoveUserFromGroup()

Remove an existing user from an existing group.

Header File: ddf.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
PRESULT DDFAPICALLTYPE PvRemoveUserFromGroup(
   BTI_WORD           dbHandle,
   const BTI_CHAR*    user);
   const BTI_CHAR*    group,
```

Arguments

In	*dbHandle*	Handle of an open database returned by PvOpenDatabase().
In	*user*	Database user name.
In	*group*	Database group name.

Return Values

PCM_Success	The operation was successful.
PCM_errFailed	The operation was not successful.
PCM_errInvalidAccountName	The specified account or user name does not exist.
PCM_errUserNotPartOfGroup	The specified user is not a member of the group.
PCM_errDatabaseHasNoSecurity	Database has no security.
PCM_errSessionSecurityError	Database opened with insufficient privilege.

Remarks

The following preconditions must be met:

- You must first open a database successfully using PvOpenDatabase() as user 'Master'.
- The associated database has database-level security enabled.

- The specified group and user names must already exist in the database.
- The specified user is a member of the specified group.

The following post condition must be met:

- Use PvCloseDatabase() to free the resources.

See Also

PvCreateGroup()
PvCreateUser()
PvAlterUserName()
PvAddUserToGroup()
PvDropGroup()
PvDropUser()
PvOpenDatabase()
PvCloseDatabase()

PvSecureDatabase()

Enables security for an existing database.

Header File: dtisecurity.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvSecureDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR_PTR      dbUser,
   BTI_CHAR_PTR      dbPassword);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the **PvConnectServer()** function.
In	*dbName*	Name of the database.
In	*dbUser*	Database user name – must be Master to set security.
In	*dbPassword*	Database password for Master user.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_FAIL	Failed to open the database for other reasons.
PCM_errSessionSecurityError	Invalid user name or password.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- When you enable security, you must specify Master as the database user name and choose a password. Security for the database is enforced based on the access rights defined for the database and should match behavior seen in SQL or ODBC access methods.

See Also

PvStart()
PvConnectServer()
PvOpenDatabase()
PvUnSecureDatabase()
PvIsDatabaseSecured()
PvCloseDatabase()
PvDisconnect()
PvStop()

PvSecureDatabase2()

Enables database security for an existing database. It differs from PvSecureDatabase() in that it supports domain authentication.

Header File: dtisecurity.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvSecureDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR_PTR      dbPassword,
   BTI_LONG          dbAuthentication);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the **PvConnectServer()** function.
In	*dbName*	Name of the database.
In	*dbPassword*	Database password for Master user.
In	*dbAuthentication*	Type of authentication to enable. Values are 1 for database and 2 for domain.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_FAIL	Failed to open the database for other reasons.
PCM_errSessionSecurityError	Invalid password.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- When you enable security, you must choose a password for the Master user. Security for the database is enforced based on the access rights defined for the database and should match behavior seen in SQL or ODBC access methods.
- You must be connecting to a Windows server. Calls to Linux or macOS servers return a general failure (status code 7004), since Active Directory domain authentication is Windows only.

See Also

PvStart()
PvConnectServer()
PvOpenDatabase()
PvSecureDatabase()
PvUnSecureDatabase()
PvIsDatabaseSecured()
PvCloseDatabase()
PvDisconnect()
PvStop()

PvSetBooleanValue()

Save new value for a Boolean type setting, to the data target specified by *whichData*.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvSetBooleanValue(
   BTI_LONG            hConnection,
   BTI_ULONG           settingID,
   BTI_SINT            newValue,
   BTI_SINT            whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In	*newValue*	Integer value to be set.
In	*whichData*	Flag to indicate which value is to be set: • PVDATA_CURRENT means apply setting changes to current session and save to registry, ini or ncf file. Only valid for Trace Op in Btr 6.15 NT release. • PVDATA_PERSISTENT don't apply setting change to the current session. Save setting to registry, ini or ncf files only.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_INVALID_DATA_TYPE	The setting is not of Boolean type.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- You must logon with administrator-level rights with PvConnectServer () before you can set a new value for a Boolean type setting.

Note: This function cannot be called by a user logged-in with the "restricted" user type.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetBooleanValue()
PvGetBooleanStrings()
PvIsSettingAvailable()
PvDisconnect()
PvStop()

PvSetLongValue()

Save new value for a long integer type setting, to the data target specified by *whichData*.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvSetLongValue(
   BTI_LONG            hConnection,
   BTI_ULONG           settingID,
   BTI_LONG            newValue,
   BTI_SINT            whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In	*newValue*	Integer value to be set. Before calling this function, check to see that the value is within the limits for the particular setting by using the PvGetValueLimit() function.
In	*whichData*	Flag to indicate which value is to be set: PVDATA_CURRENT sets current value. PVDATA_PERSISTENT sets persistent value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_INVALID_DATA_TYPE	The setting is not of long type.
P_E_OUT_OF_RANGE	The value specified to be set is out of range.
P_E_FAIL	Failed for other reasons.

Remarks

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- You must logon with administrator-level rights with PvConnectServer () before you can set a new value for a Long type setting.

Note: This function cannot be called by a user logged-in with the "restricted" user type.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetLongValue()
PvGetValueLimit()
PvIsSettingAvailable()
PvDisconnect()
PvStop()

PvSetSelectionValue()

Save new value for a selection type setting, to the data target specified by *whichData*.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvSetSelectionValue(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_ULONG         numItems,
   BTI_LONG_PTR      pNewValue,
   BTI_SINT          whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In	*numItems*	Number of individual selection items to be set.
In	*pNewValue*	Array of individual selection items to be set.
In	*whichData*	Flag to indicate which value is to be set: PVDATA_CURRENT sets current value. PVDATA_PERSISTENT sets persistent value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The setting is not of selection type.
P_E_INVALID_SELECTION	At least one selection item is invalid.
P_E_FAIL	Failed for other reasons.

Remarks

This function is used to work with both single-selection and multi-selection data types. If more than one selection items are set for a single-selection item, the first value is used.

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- You must logon with administrator-level rights with PvConnectServer () before you can set a new value for a Selection type setting.

Note: This function cannot be called by a user logged-in with the "restricted" user type.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetSelectionValue()
PvGetSelectionString()
PvGetAllPossibleSelections()
PvCountSelectionItems()
PvIsSettingAvailable()
PvDisconnect()
PvStop()

PvSetStringValue()

Save new value for a string type setting, to the data target specified by *whichData*.

Header File: config.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvSetStringValue(
   BTI_LONG          hConnection,
   BTI_ULONG         settingID,
   BTI_CHAR_PTR      newValue,
   BTI_SINT          whichData);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.
In	*settingID*	Unique identifier for the setting. A list of settings can be obtained from PvGetSettingList().
In	*newValue*	String value to be set.
In	*whichData*	Flag to indicate which value is to be set: PVDATA_CURRENT sets current value. PVDATA_PERSISTENT sets persistent value

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_INVALID_DATA_TYPE	The setting is not of string type.
P_E_FAIL	Failed for other reasons.

Remarks

Some settings may take multiple strings separated by semicolons (;).

The following preconditions must be met:

- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- You must logon with administrator-level rights with PvConnectServer() before you can set a new value for a String type setting.

Note: This function cannot be called by a user logged-in with the "restricted" user type.

See Also

PvStart()
PvConnectServer()
PvGetCategoryList()
PvGetSettingList()
PvGetStringType()
PvGetStringValue()
PvIsSettingAvailable()
PvDisconnect()
PvStop()

PvStart()

Start a Distributed Tuning Interface (DTI) session. This function must be called before any DTI calls are made.

Header File: connect.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvStart(BTI_LONG reserved);
```

Arguments

In	*reserved*	Reserved for future use.

Return Values

P_OK	The operation was successful.
P_E_FAIL	A general failure occurred.

Remarks

This function performs initialization and binds resources for DTI.

Example

```
BTI_SINT status = 0;

status = PvStart(0);
// invoke multiple DTI calls
status = PvStop (0);
```

See Also

PvStop()

PvStop()

Closes a DTI session and frees the related resources.

Header File: connect.h (See also Header Files)

Function First Available In Library: w3dbav75.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_SINT PvStop(BTI_LONG_PTR preserved);
```

Arguments

In	*preserved*	Reserved for future use.

Return Values

P_OK	The operation was successful.
P_E_FAIL	A general failure occurred.

Remarks

This function frees resources of DTI and closes the DTI session. This function should be called before your application exits.

Example

```
BTI_LONG status = 0;

status = PvStop(0);
```

See Also

PvStart()

PvUnSecureDatabase()

Disables database security on a database.

Header File: dtisecurity.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvUnSecureDatabase(
   BTI_LONG          hConnection,
   BTI_CHAR_PTR      dbName,
   BTI_CHAR_PTR      dbUser,
   BTI_CHAR_PTR      dbPassword);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the **PvConnectServer()** function.
In	*dbName*	Name of the database.
In	*dbUser*	Database user name - must be Master to enable or disable security.
In	*dbPassword*	Database password for Master user.

Return Values

P_OK	The operation was successful.
P_E_INVALID_HANDLE	Invalid connection handle.
P_E_NULL_PTR	Call with NULL pointer.
P_E_ACCESS_RIGHT	Insufficient access right for the operation.
P_E_FAIL	Failed to open the database for other reasons.
PCM_errSessionSecurityError	Invalid user name or password.

Remarks

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.
- Database is secured.

See Also

PvStart()
PvConnectServer()
PvOpenDatabase()
PvSecureDatabase()
PvIsDatabaseSecured()
PvCloseDatabase()
PvDisconnect()
PvStop()

PvValidateLicenses()

Initiates a check of the validity of all keys on the computer indicated by the connection.

Header File: dtilicense.h (See also Header Files)

Function First Available In Library: w3dbav90.dll (Windows), libpsqldti.so (Linux), libpsqldti.dylib (macOS) (See also Link Libraries)

Syntax

```
BTI_API PvValidateLicenses(BTI_LONG hConnection);
```

Arguments

In	*hConnection*	Connection handle that identifies the server. Connection handles are obtained with the PvConnectServer() function.

Return Values

P_OK	The validation operation completed successfully.
P_E_FAIL	The validation operation did not complete successfully.
Status code pertaining to license administration or to authorization	See *Status Codes and Messages* for License Administrator Status Codes and Authorization Status Codes.

Remarks

PvValidateLicenses returns only the result from *requesting* a validation check. It does *not* return any information about the state of the keys. You must separately call PvGetProductsInfo() to get the XML string of product information that includes information about the state of the keys.

The following preconditions must be met:

- DTI session started by calling PvStart().
- Connection established by PvConnectServer() or if you are performing the operation on a local machine, P_LOCAL_DB_CONNECTION may be used as the connection handle.

Example

```
status = PvValidateLicenses(P_LOCAL_DB_CONNECTION);
```

See Also

PvGetProductsInfo()
PvStart()
PvStop()