Contents

Installation

What You Will Need: Environment Information
Instructions: How to Install
Files Created and Affected by Install

Application Architecture

Sample Application

OverView

Asychronous Programming
Callbacks
Capabilities
CT-Library Messages
Dynamic SQL
Error and Message Handling
Results
Types

Function Descriptions

CompileDate
vbcs_calc
vbcs_cmp
vbcs_config
vbcs_convert
vbcs_ctx_alloc
vbcs_ctx_drop
vbcs_ctx_global
vbcs_decode_message
vbcs_diag
vbcs_dt_crack
vbcs_dt_info
vbcs_get_client_msg
vbcs_get_server_msg
vbcs_loc_alloc
vbcs_loc_drop

vbcs_locale

vbcs_objects

vbcs_setnull

vbcs_strcmp

vbcs_time

vbcs_will_convert

vbct_cancel

vbct_capability

vbct_close

vbct_cmd_alloc

vbct_cmd_drop

vbct_cmd_props_num

vbct_cmd_props_str

vbct_command_num

vbct_command_str

vbct_con_alloc

vbct_con_drop

vbct_con_props_num

vbct_con_props_str

vbct_compute_info

vbct_config_num

vbct_config_str

vbct_connect

vbct_data

vbct_data_info

vbct_deinstall_callbacks

vbct_describe

vbct_diag

vbct_do_binds

vbct_dynamic

vbct_dyndesc

vbct_exit

vbct_fetch

vbct_get_data

vbct_getformat

vbct_getloginfo

vbct_init

vbct_install_callbacks
vbct_labels
vbct_options_num
vbct_options_str
vbct_param
vbct_poll
vbct_recvpassthru
vbct_remote_pwd
vbct_res_info
vbct_results
vbct_send
vbct_send_data
vbct_sendpassthru
vbct_setloginfo
vbct_wakeup
vbblk_alloc
vbblk_bind
vbblk_describe
vbblk_done
vbblk_get_data
vbblk_init
vbblk_props_num
vbblk_props_str
vbblk_set_data
vbblk_rowxfer
vbblk_textxfer

What You Will Need: Environment Information

To use SQL-Sombrero/VBX you will need the following:

- Microsoft Windows   release 3.1 or later.
- 80386 (20MHz) or higher processor.
- 4 megabytes of RAM.
- 2 megabytes of free disk space for softwre and enviromment.

Need to put trademarks in.

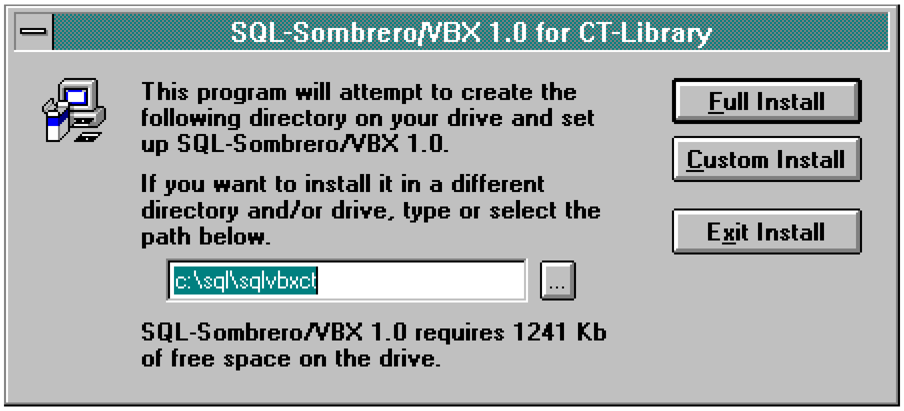
- SYBASE SQL Server System 10 or greater Open Client   software (runtime CT-Library and Net-Library) for Windows and compatible network.

Instructions: How to Install

SQL-Sombrero/VBX comes with one install diskette.

Insert the Install diskette and run INSTALL.EXE from within windows.

The default is a directory called C:\SQL\SQLVBXCT. A drop-down list box facilitates choosing an alternative destination.



SQL-Sombrero/VBX 1.0 for CT-Library

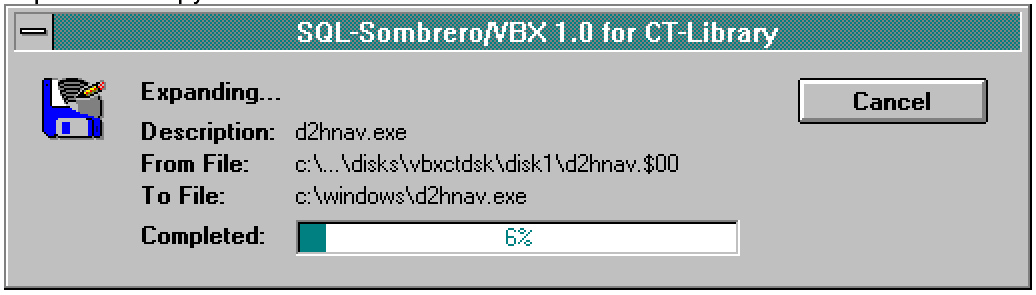
This program will attempt to create the following directory on your drive and set up SQL-Sombrero/VBX 1.0.

If you want to install it in a different directory and/or drive, type or select the path below.

c:\sql\sqlvbxct

SQL-Sombrero/VBX 1.0 requires 1241 Kb of free space on the drive.

FULL INSTALL
Press the Full Install button to install SQL-Sombrero/VBX completely. The installation process begins to unpack and copy files from the Install diskette to the chosen destination.

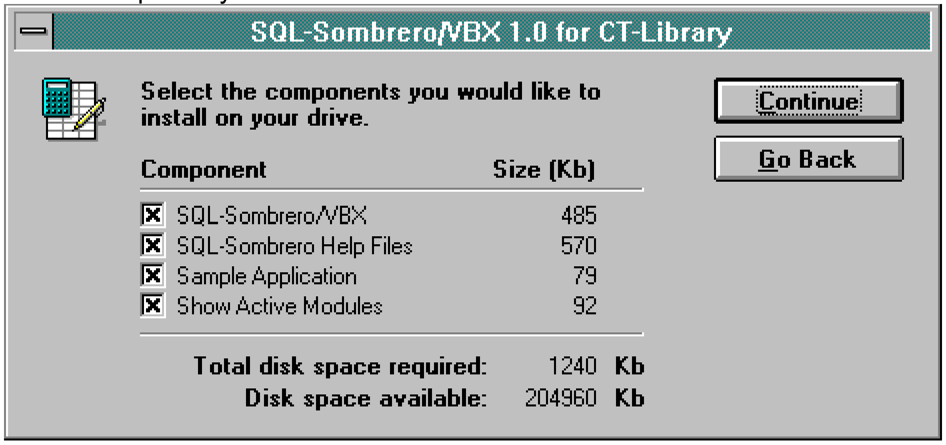


SQL-Sombrero/VBX 1.0 for CT-Library

Expanding...

Description: d2hnav.exe
From File: c:\...\disks\vbxctdsk\disk1\d2hnav.$00
To File: c:\windows\d2hnav.exe
Completed: 6%

CUSTOM INSTALL
The Custom install procedure allows individual components of the SQL-Sombrero/VBX package to be installed separately.



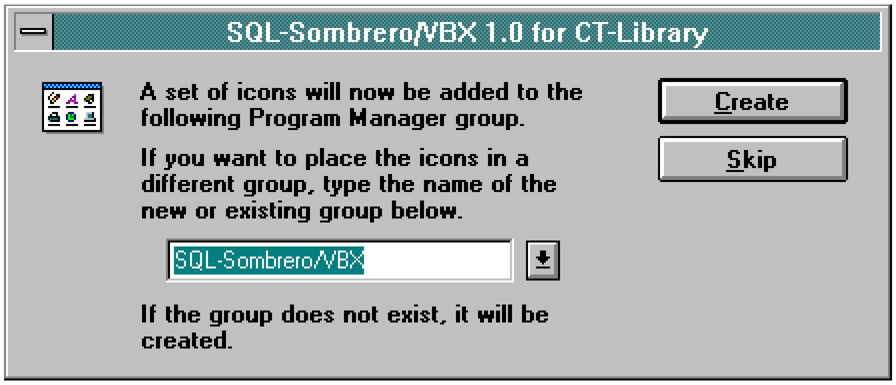
SQL-Sombrero/VBX 1.0 for CT-Library

Select the components you would like to install on your drive.

Component	Size (Kb)
☒ SQL-Sombrero/VBX	485
☒ SQL-Sombrero Help Files	570
☒ Sample Application	79
☒ Show Active Modules	92

Total disk space required: 1240 Kb
Disk space available: 204960 Kb

NOTE: It is important that the latest DLLs are used in order to achieve maximum functionality.
When the installation of files is complete, the install procedure will ask you if you wish to create a Program Manager group to access the help files and the Showmod program..


SQL-Sombrero/VBX 1.0 for CT-Library

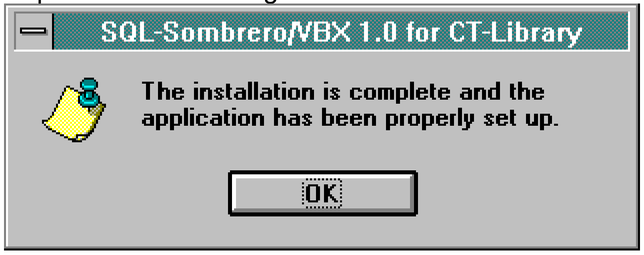
A set of icons will now be added to the following Program Manager group.

If you want to place the icons in a different group, type the name of the new or existing group below.

SQL-Sombrero/VBX

If the group does not exist, it will be created.

After creating the Program Manager group the install will inform you that the installation is complete and request an acknowledgement.


SQL-Sombrero/VBX 1.0 for CT-Library

The installation is complete and the application has been properly set up.

Files Created and Affected by Install

If you choose the Custom installation option there are a total of four (4) different components that can be installed.

1. SQL-Sombrero/VBX

This component consists of the VBX file with its license file and the files containing the declarations for the SQL-Sombrero/VBX functions and declaration of CT-Library constants and user defined variable types.

If this component is selected the following files are installed. The files are installed to the directory chosen in the installation procedure as described above.

SQLVBXCT.VBX

SQLDEF.BAS

CSCONST.BAS

CSTYPES.BAS

CTCONST.BAS

SQLVBXCT.LIC

README.TXT

INSTALL.LOG (Shows files installed and their locations)

2. SQL-Sombrero/VBX Help File

This component consists of the Help file for SQL-Sombrero/VBX as well as other files to allow the navigator option to work with the help file.

If this component is selected the following files are installed. The files are installed to the directories as shown below.

SQLVBXCT.HLP	- default directory
SQLVBXCT.DHN	- default directory
D2HNAV.EXE	- windows directory
D2HNAV.HLP	- windows directory
MSOUTLIN.VBX	- windows system directory
D2HLINK.DLL	- windows system directory

3. Sample Application

This is a sample application using the SQL-Sombrero/VBX for SYBASE SQL Server. This sample requires the PUBS2 database to be installed.

LOGON.FRM	- default/sample 10 directory
MAINFORM.FRM	- default/sample 10 directory
GBASS.BAS	- default/sample 10 directory
SAMPLE10.MAK	- default/sample 10 directory

SQLDEF.BAS	- default/sample 10 directory
CSCONST.BAS	- default/sample 10 directory
CSTYPES.BAS	- default/sample 10 directory
CTCONST.BAS	- default/sample 10 directory

4. Show Active Modules

This is a program that will scan all modules currently running in the Windows environment and show details of the modules.

SHOWMOD.EXE	- default directory

The file SQLVBXCT.LIC is the license file. This file must reside in the same directory as the file SQLVBXCT.VBX. This file is required if the SQL-Sombrero/VBX for CT-Library is to be used in developing an application. The SQLVBXCT.LIC file is not required for an application that is distributed as an EXE file.

Please refer to the inside of this manual's front cover for more information regarding the LIC file.

Sample Application

Setting up and installing callbacks
Installing Message Callback Routines
Connecting to a Server
Changing databases
Sending a command to the server
Processing the results of the command
Finishing up

Setting up and installing callbacks

The CT-Library routine vbcs_ctx_alloc allocates a context structure.   A context structure is used to store configuration parameters that describe a particular "context," or operating environment, for a set of server connections.   On most platforms, an application can have multiple contexts, although a typical application will need just one.

Application properties that can be defined at the context level include the name and location of the interfaces file, the login timeout value, and the maximum number of connections allowed within the context.

vbct_init initializes CT-Library.   An application calls vbct_init after calling vbcs_ctx_alloc and before calling any other CT-Library routine.

Installing Message Callback Routines

vbct_install_callbacks installs a CT-Library callback routine. Callbacks are custom routines which are called automatically by CT-Library when a triggering event of the appropriate type occurs. For example, a client message callback is called automatically whenever CT-Library generates an error or informational message.

There are several types of callbacks, but the example program installs only two: a client message callback, to handle CT-Library error and informational messages, and a server message callback, to handle server error and informational messages.

```
If ctx_pointer = 0 Then

        ret = VBCS_ctx_alloc(CS_VERSION_100, ctx_pointer)

        ret = VBCT_init(ctx_pointer, CS_VERSION_100)

        ret = VBCT_install_callbacks(ctx_pointer,CS_SERVERMSG_CB)

        ret = VBCT_install_callbacks(ctx_pointer, CS_CLIENTMSG_CB)

End If
```

Connecting to a Server

vbct_con_alloc allocates a connection structure.   A connection structure contains information about a particular client/server connection.

vbct_con_props_num and vbct_con_props_str sets and retrieves the values of a connection's properties. Connection properties include user name and password, which are used in logging into a server; application name, which appears in SQL Server's sysprocess table, and packet size, which determines the size of network packets that an application will send and receive.

vbct_connect opens a connection to a server, logging into the server with the connection information specified via vbct_con_props_num and vbct_con_props_str.

The variables sbuf, pword are strings containing the userid and password obtained from a logon form.

```
sbuf = uid

ret = VBCT_con_alloc(ctx_pointer, con_pointer)

ret = VBCT_con_props_str(con_pointer, CS_SET, CS_USERNAME, sbuf, outlen)

If pword <> "" Then

        ret = VBCT_con_props_str(con_pointer, CS_SET, CS_PASSWORD, pword, outlen)

End If

ret = VBCT_con_props_str(con_pointer, CS_SET, CS_APPNAME, "TestCTLib", outlen)

ret = VBCT_con_props_str(con_pointer, CS_SET, CS_HOSTNAME, "TestCTLib", outlen)

ret = VBCT_connect(con_pointer, "")
```

Changing databases

CT-Library requires that a command be sent to change the database. The sample application requires data from the pubs2 database. The following code fragment changes the database to pubs2. It allocates a command structure for use in changing the database. This command structure is used to process all commands within the sample application.

```
ret = VBCT_cmd_alloc(con_pointer, cmd_pointer)

ret = process_no_rows("use pubs2")
```

Sending a command to the server

vbct_cmd_alloc allocates a command structure.   A command structure is used to send commands to a server and to process the results of those commands.

vbct_command initiates the process of sending a non-cursor command.   In this case, the example program initiates a language command.

vbct_send sends a command to the server.

```
sql$="select 'Author Id' = au_id , 'Last Name' = au_lname , 'First Name' = au_fname   from authors"

ret = VBCT_command_str(cmd_pointer, CS_LANG_CMD, sql$, CS_UNUSED)

ret = VBCT_send(cmd_pointer)
```

Processing the results of the command

Almost all CT-Library programs will process results by using a loop controlled by vbct_results. Inside the loop, a switch takes place on the current type of result. Different types of results require different types of processing.

An application can call vbct_res_info to get the number of result columns. The application will call vbct_do_binds to bind the results to SQL-Sombrero internal program variables. After the result items are bound, vbct_fetch is called to fetch data rows until end-of-data. The application will retrieve the data from the columns using vbct_data.

```
result_ret = VBCT_results(cmd_pointer, restype)

res$ = ""

While result_ret = CS_SUCCEED

        Select Case restype

        Case CS_ROW_RESULT

                ret = VBCT_res_info(cmd_pointer, CS_NUMDATA, ncols)

                ret = VBCT_do_binds(cmd_pointer)

                While ret = CS_SUCCEED Or ret = CS_ROW_FAIL

                        ret = VBCT_fetch(cmd_pointer, 0, 0, 0, rows_read)

                        If ret = CS_SUCCEED Then

                                For i = 1 To ncols

                                        t(i) = VBCT_data(cmd_pointer, i)

                                Next

                        End If

                Wend

        Case CS_CMD_DONE, CS_CMD_SUCCEED, CS_CMD_FAIL

        Select Case restype

        Case CS_CMD_DONE

                rt$ = "CS_CMD_DONE"

        Case CS_CMD_SUCCEED

                rt$ = "CS_CMD_SUCCEED"

        Case CS_CMD_FAIL

                rt$ = "CS_CMD_FAIL"

        End Select

        Case Else

        End Select

        result_ret = VBCT_results(cmd_pointer, restype)
```

Wend

Finishing up

vbct_cmd_drop de-allocates a command structure.

vbct_close closes a server connection.

vbct_con_drop de-allocates a connection structure.

vbct_exit terminates CT-Library.

The CT-Library routine vbcs_ctx_drop de-allocates a context structure.

```
ret = VBCT_cmd_drop(cmd_pointer)

ret = VBCT_close(con_pointer, CS_FORCE_CLOSE)

ret = VBCT_con_drop(con_pointer)

ret = VBCT_exit(ctx_pointer, CS_FORCE_EXIT)

ret = VBCS_ctx_drop(ctx_pointer)
```

Asychronous Programming

Asynchronous applications are designed to make constructive use of time that would otherwise be spent waiting for certain types of operations to complete.   Typically, reading from and writing to a network or external device is much slower than straightforward program execution.

When writing an asynchronous application, the application programmer must enable asynchronous CT-Library behavior at the context or connection level by setting the CT-Library property CS_NETIO to CS_ASYNC_IO.   When asynchronous behavior is enabled, all CT-Library routines that read from or write to the network either:

Initiate the requested operation and return CS_PENDING immediately

Return CS_BUSY to indicate that an asynchronous operation is already pending for this connection.

Non-asynchronous routines can also return CS_BUSY if called when an asynchronous operation is pending for a connection.

Related Topics:

Learning about Completions

Asynchronous Routines

CT-Library's Interrupt-Level Memory Requirements

Browse Mode

Implementing Browse Mode

Browse-mode Conditions

Learning about Completions

An application can learn of an asynchronous routine completion in one of two ways:

- CT-Library automatically calls the application's completion callback routine when an asynchronous operation completes.
- On platforms that do not support interrupt-driven I/O, an application can use vbct_poll to find out if any asynchronous operations have completed. If it finds a completed operation, CT-Library will call an application's completion callback routine from within vbct_poll.

Asynchronous Routines

The following CT-Library routines can behave asynchronously:

vbct_cancel

vbct_close

vbct_connect

vbct_fetch

vbct_get_data

vbct_options_num

vbct_options_str

vbct_recvpassthru

vbct_results

vbct_send

vbct_send_data

vbct_sendpassthru

Any CT-Library routine that takes a command or connection structure as a parameter can return CS_BUSY. CS_BUSY indicates that a routine is unable to perform because the relevant connection is currently busy, waiting for an asynchronous operation to complete.

An application can call the following routines while an asynchronous operation is pending:

Any routine that takes a CS_CONTEXT structure as a parameter. If the CS_CONTEXT structure is an optional parameter, it must be non-NULL.

vbct_cancel(CS_CANCEL_ATTN)

vbct_cmd_props_num(CS_USERDATA)

vbct_cmd_props_str(CS_USERDATA)

vbct_con_props_num(CS_USERDATA)

vbct_con_props_str(CS_USERDATA)

vbct_poll

CT-Library's Interrupt-Level Memory Requirements

Ordinarily, CT-Library routines satisfy their memory requirements by calling malloc. However, because not all implementations of malloc are re-entrant, it is not safe for CT-Library routines that are called at the interrupt level to use malloc. For this reason, asynchronous applications are required to provide an alternate way for CT-Library to satisfy its memory requirements.

CT-Library provides the following two mechanisms by which an asynchronous application can satisfy CT-Library's memory requirements:

- The application can use the CS_MEM_POOL property to provide CT-Library with a memory pool.
- The application can use the CS_USER_ALLOC and CS_USER_FREE properties to install memory allocation routines that CT-Library can safely call at the interrupt level.

If an asynchronous application fails to provide CT-Library with a safe way to satisfy memory requirements, CT-Library's behavior is undefined.

CT-Library attempts to satisfy memory requirements from the following sources in the following order:

1. Memory pool
2. User-supplied allocation and free routines
3. System routines

Browse Mode

NOTE: Browse mode is included in 10.0 CT-Library in order to provide compatibility with Open Server applications and older Open Client libraries. Its use in new Open Client CT-Library applications is discouraged, because cursors provide the same functionality in a more portable and flexible manner. Further, browse mode is SYBASE-specific and is not suited for use in a heterogeneous environment.

Browse mode provides a means for browsing through database rows and updating their values a row at a time. From the standpoint of an application program, the process involves several steps, because each row must be transferred from the database into program variables before it can be browsed and updated.

Since a row being browsed is not the actual row residing in the database, but is instead a copy residing in program variables, the program must be able to ensure that changes to the variables' values can be reliably used to update the original database row. In particular, in multi-user situations, the program needs to ensure that updates made to the database by one user do not unwittingly overwrite updates recently made by another user. This can be a problem because an application typically selects a number of rows from a database at one time, but the application's users browse and update the database one row at a time. A timestamp column in browsable tables provides the information necessary to regulate this type of multi-user updating.

Because some applications permit users to enter ad-hoc browse mode queries, CT-Library provides two routines, ct_br_table and ct_br_column, that allow an application to retrieve information about the tables and columns underlying a browse-mode result set. This information is useful when an application is constructing commands to perform browse-mode updates.

A browse-mode application requires two connections, one for selecting the data and a second for performing the updates.

Implementing Browse Mode

Conceptually, browse mode involves the following two steps:

1. Select rows containing columns derived from one or more database tables.

2. Where appropriate, change values in columns of the result rows (not the actual database rows), one row at a time, and use the new values to update the original database tables.

The following steps are implemented in a program:

1. Set a connection's CS_HIDDEN_KEYS property to CS_TRUE. This ensures that CT-Library returns a table's timestamp column as part of a result set. In browse-mode updates, the timestamp column is used to regulate multi-user updates.

2. Execute a select...for browse language command. This command generates a regular row result set. This result set contains hidden key columns (one of which is the timestamp column) in addition to explicitly selected columns.

3. After vbct_results indicates regular row results, call vbct_describe to get CS_DATAFMT descriptions of the result columns:

- To indicate the timestamp column, vbct_describe sets the CS_TIMESTAMP and CS_HIDDEN bits in the datafmt.status field.

- To indicate an ordinary hidden key column, vbct_describe sets the CS_HIDDEN bit in the datafmt.status field. If the CS_HIDDEN bit is not set, the column is an explicitly-selected column.

4. Call vbct_do_binds to bind the result columns of interest. An application must bind all hidden columns because it will need the values of these columns to build a qualifier at update time.

5. Call ct_br_table , if necessary, to retrieve information about the database tables that underlie the result set. Call ct_br_column, if necessary, to retrieve information about a specific result set column. Both of these types of information can be useful when building a language command to update the database.

6. Call vbct_fetch in a loop to fetch rows. When a row is fetched that contains values that need to be changed, update the database table(s) with the new values. Complete the following to carry out this procedure:

- Construct a language command containing a Transact-SQL update statement with a where-clause qualifier that uses the row's hidden columns (including the timestamp column).

- Send the language command to the server and process the results of the command.

A language command containing a browse-mode update statement generates a result set of type CS_PARAM_RESULT. This result set contains a single result item, the new timestamp for the row.

If the application plans to update this same row again, it must save the new timestamp for later use.

After one browse-mode row has been updated, the application can fetch and process the next row.

Browse-mode Conditions

To use browse mode, the following conditions must be true:

- The select command that generated the result set must end with the key words for browse.
- The table(s) to be updated must be "browsable" (i.e., each must have a unique index and a timestamp column).
- The result columns to be updated cannot be the result of SQL expressions, such as max(colname).

Callbacks

What Are Callbacks?
When Are Callbacks Called?
Types of Callbacks
Installing a Callback Routine
When a Callback Event Occurs
Defining Callback Routines
Client Message Callbacks
Client Message Callback Information
Completion Callbacks
Completion Callback Information
Notification Callbacks
Defining a Notification Callback
Server Message Callbacks
Defining a Server Message Callback

What Are Callbacks?

Callbacks are user-supplied routines that are automatically called by CT-Library whenever certain triggering events, known as callback events, occur.

Some callback events are the result of a server response arriving for an application. For example, a notification callback event occurs when a registered procedure notification arrives from an Open Server.

Other callback events occur at the internal CT-Library level. For example, a client message callback event occurs when CT-Library generates an error message.

When Are Callbacks Called?

When CT-Library recognizes a callback event, it automatically calls the appropriate callback routine.

In order for CT-Library to recognize some callback events, it must be actively engaged in reading from the network. Most callback events of this type occur when CT-Library is naturally reading from the network, and are handled automatically.

Two types of callback events, however, can occur when CT-Library is not reading from the network. They are as follows:

- The completion callback event, which occurs when an asynchronous CT-Library routine completes.
- The notification callback event, which occurs when an Open Server notification arrives for an application.

If a platform supports interrupt-driven I/O, completion and notification callbacks are called at the interrupt level when a completion or notification callback event occurs. If a platform does not support interrupt-driven I/O, however, an application will need to call vbct_poll to check for these events if it is not otherwise reading from the network.

If vbct_poll finds an asynchronous routine completion or an Open Server notification, it automatically calls the appropriate callback routine before returning.

NOTE: Because some types of callback routines can be executed at interrupt time, a callback routine must take care in accessing data structures that are also used by the application's main-line code.

Types of Callbacks

The following table lists the types of callbacks, when they are called, and whether an application needs to use vbct_poll to trigger them:

Type of Callback:	When Is it Called?	How Is it Called?
Client Message	In response to a CT-Library error or informational message.	When CT-Library generates an error or informational message, CT-Library automatically triggers the client message callback.
Completion	When an asynchronous CT-Library routine completes.	An asynchronous routine completion can occur at any time. On platforms that support interrupt-driven I/O, the completion callback is called automatically, at the interrupt level, when the completion occurs. On platforms that do not support interrupt-driven I/O, an application can use vbct_poll to find out if any routines have completed.
Notification	When an Open Server notification arrives.	An Open Server notification can arrive at any time. On platforms that support interrupt-driven I/O, the notification callback is called at the interrupt level when the completion occurs. On platforms that do not support interrupt-driven I/O, an application must be actively reading from the network in order for CT-Library to recognize a notification. If an application is not actively reading from the network, it can use vbct_poll to find out if a notification has arrived.
Server Message	In response to a server error or informational message.	Server messages occur as the result of specific commands. When an application processes the results of a command, CT-Library reads any error or informational messages related to the command, automatically triggering the server message callback.

Installing a Callback Routine

An application installs a callback routine by calling vbct_install_callbacks, passing a pointer to the callback routine and indicating its type via the type parameter.

A callback of a particular type is installed at the context level. When a connection is allocated, it picks up default callbacks from its parent context.

When a Callback Event Occurs

For most types of callbacks, when a callback event occurs:

- If a callback of the proper type exists at the proper level, it is called.
- If a callback of the proper type does not exist at the proper level then the callback event information is discarded.

The client message callback is an exception to this rule. When an error or informational message is generated for a connection that has no client message callback installed, CT-Library calls the connection's parent context's client message callback (if any) rather than discarding the message. If the context has no client message callback installed, then the message is discarded.

Defining Callback Routines

All callback routines are limited as to which CT-Library routines they can call.   The following table lists types of callback routines and the CT-Library routines that they can call:

Type of Callback	Can Call	Under What Circumstances?
All Callback Routines	vbct_config_num, vbct_config_str	To retrieve information only.
	vbct_con_props_num vbct_con_props_str	To retrieve information or to set the CS_USERDATA property only.
	vbct_cmd_props_num vbct_cmd_props_str	To retrieve information or to set the CS_USERDATA property only.
	vbct_cancel (CS_CANCEL_ATTN)	
Server Message	vbct_do_binds, vbct_describe, vbct_fetch, vbct_get_data, vbct_res_info	The routines must be called with the command structure returned by the callbacks's vbct_con_props_num(CS_EED_CMD) call.
Notification	vbct_do_binds, vbct_describe, vbct_fetch, vbct_get_data, vbct_res_info(CS_NUMDATA)	The routines must be called with the command structure returned by the callbacks's vbct_con_props_num (CS_NOTIF_CMD) call.
Completion	Any CT-Library except vbcs_objects (CS_SET), vbct_init, vbct_exit, vbct_setloginfo, and vbct_getloginfo. vbcs_objects (CS_SET) is not asynchronous-safe, and vbct_init, vbct_exit, vbct_setloginfo, and vbct_getloginfo perform system-level memory allocation or free.	

The following sections contain information on the information return in each type of callback.

Client Message Callbacks

An application can handle CT-Library error and informational messages in-line, or through a client message callback routine.

When a connection is allocated, it picks up a default client message callback from its parent context. If the parent context has no client message callback installed, then the connection is created without a default client message callback.

After allocating a connection, an application can do the following:

- Install a different client message callback for the connection.
- Call vbct_diag to initialize in-line message handling for the connection. Note that vbct_diag automatically de-installs all message callbacks for the connection.

If a client message callback is not installed for a connection or its parent context and in-line message handling is not enabled, CT-Library discards message information.

If callbacks are not implemented for a particular programming language/platform version of CT-Library, an application must handle CT-Library messages in-line, using vbct_diag.

If a connection is handling CT-Library messages through a client message callback, then the callback is called whenever CT-Library generates an error or informational message.

NOTE: The exception to this rule is that CT-Library does not call the client message callback when a message is generated from within most types of callback routines. CT-Library does call the client message callback when a message is generated within a completion callback.

This means if a CT-Library routine fails within a callback routine other than the completion callback, the routine returns CS_FAIL but does not trigger the client message callback.

Client Message Callback Information

Sub SQLCTLIB1_CLIENTMESSAGE (Context As Long, Connection As Long)

Where:

Context is a pointer to the CS_CONTEXT structure for which the message occurred.

Connection is a pointer to the CS_CONNECTION structure for which the message occurred, if any. connection can be NULL.

The callback uses the following code to obtain the message information.

```
Dim cmsg As CS_CLIENTMSG

Dim ret As Long

Dim layer As Long

Dim origin As Long

Dim severity   As Long

Dim number As Long

crlf$ = Chr(13) & Chr(10)

ret = VBCS_get_client_msg(cmsg)

VBCS_decode_message cmsg.msgnumber, layer, origin, severity, number
```

The following table lists the CT-Library routines that a client message callback can call:

A Client Message Callback Can Call:	**Under What Circumstances?**
vbct_config_num vbct_config_str	To retrieve information only.
vbct_con_props_num vbct_con_props_str	To retrieve information or to set the CS_USERDATA property only.
vbct_cmd_props_num vbct_cmd_prop_str	To retrieve information or to set the CS_USERDATA property only.
vbct_cancel (CS_CANCEL_ATTN)	Any.

Completion Callbacks

A completion callback is called whenever an application receives notice that an asynchronous routine has completed.

A context or a connection can be defined to be asynchronous.   If a context is asynchronous, then all connections within that context are asynchronous, unless defined otherwise.

When a connection is asynchronous, CT-Library routines that perform network I/O do not block, but instead return CS_PENDING immediately.   When a routine completes, CT-Library automatically calls the completion callback.

A completion callback is typically coded to notify the main-line code of the asynchronous routine's completion.

Completion Callback Information

A completion callback is defined as follows:

Sub SQLCTLIB1_COMPLEMESSAGE (Connection As Long, Cmnd As Long, CFunction As Long, FStatus As Long)

where:

Connection is a pointer to the CS_CONNECTION structure representing the connection that performed the I/O for the routine.

Cmnd is a pointer to the CS_COMMAND structure for the routine, if any. cmd can be NULL.

Cfunction indicates which routine has completed.   The following table lists the symbolic values possible for Cfunction:

Value of Cfunction:	**Indicating:**
BLK_ROWXFER	blk_rowxfer has completed.
BLK_SENDROW	blk_sendrow has completed.
BLK_SENDTEXT	blk_sendtext has completed.
BLK_TEXTXFER	blk_textxfer has completed
CT_CANCEL	ct_cancel has completed.
CT_CLOSE	ct_close has completed.
CT_CONNECT	ct_connect has completed.
CT_FETCH	ct_fetch has completed.
CT_GET_DAT	Act_get_data has completed.
CT_OPTIONS	ct_options has completed.
CT_RECVPASSTHRU	ct_recvpassthru has completed.
CT_RESULTS	ct_results has completed.
CT_SEND	ct_send has completed.
CT_SEND_DATA	ct_send_data has completed.
CT_SENDPASSTHRU	ct_sendpassthru has completed.
A user-defined value. This value must be greater than or equal to CT_USER_FUNC	A user-defined function has completed.

Fstatus is the return status of the completed routine.   To find out what values status can have, refer to **Results** in the manual for the routine(s).

Because it is regarded as an extension of main-line code, a completion callback can call any CT-Library routine.

Notification Callbacks

A registered procedure is a type of procedure that is defined and installed in a running Open Server. A CT-Library application can use a remote procedure call command to execute a registered procedure, and can "watch" for a registered procedure to execute.

When a registered procedure executes, applications watching for it receive a notification that includes the procedure's name and the arguments it was called with.

When CT-Library receives a notification, it calls an application's notification callback routine.

The registered procedure's name is available as the second parameter to the notification callback routine.

The arguments with which the registered procedure was called are available inside the notification callback, as a parameter result set. To retrieve these arguments, an application:

- Calls vbct_con_props_num (CS_NOTIF_CMD) to retrieve a pointer to the command structure containing the parameter result set.
- Calls vbct_res_info(CS_NUMDATA), vbct_describe, vbct_do_binds, vbct_fetch, and vbct_get_data to describe, bind, and fetch the parameters.

Defining a Notification Callback

A notification callback is defined as follows:

Sub SQLCTLIB1_NOTIFICATION (Connection As Long, ProcName As String, ProcNameLen As Long)

where:

Connection is a pointer to the CS_CONNECTION structure receiving the notification.

ProcName is the name of the registered procedure that has been executed.

ProcNameLen is the length in bytes of ProcName

The following table lists the CT-Library routines that a notification callback can call:

A Notification Callback Can Call:	**Under What Circumstances?**
vbct_config_num vbct_config_str	To retrieve information only.
vbct_con_props_num vbct_con_props_str	To retrieve information or to set the CS_USERDATA property only.
vbct_cmd_props_num vbct_cmd_props_str	To retrieve information or to set the CS_USERDATA property only.
vbct_cancel(CS_CANCEL_ATTN)	Any
vbct_do_binds, vbct_describe, vbct_fetch, vbct_data, vbct_res_info(CS_NUMDATA)	The routines must be called with the command structure returned by the callbacks's vbct_con_props_num(CS_NOTIF_CMD) call.

Server Message Callbacks

An application can handle server error and informational messages in-line, or through a server message callback routine.

When a connection is allocated, it picks up a default server message callback from its parent context. If the parent context has no server message callback installed, then the connection is created without a default server message callback.

After allocating a connection, an application can:

- Install a different server message callback for the connection.
- Call vbct_diag to initialize in-line message handling for the connection. Note that vbct_diag automatically de-installs all message callbacks for the connection.

If a server message callback is not installed and in-line message handling is not enabled, CT-Library discards server message information.

If a connection is handling server messages through a server message callback, then the callback is called whenever a server message arrives.

Defining a Server Message Callback

A server message callback is defined as follows:

Sub SQLCTLIB1_SERVERMESSAGE (Context As Long, Connection As Long)

Where:

Context is a pointer to the CS_CONTEXT structure for which the message occurred.

Connection is a pointer to the CS_CONNECTION structure for which the message occurred, if any. connection can be NULL.

The callback uses the following code to obtain the message information.

```
Dim cmsg As CS_SERVERMSG

Dim ret As Long

Dim layer As Long

Dim origin As Long

Dim severity   As Long

Dim number As Long

crlf$ = Chr(13) & Chr(10)

ret = VBCS_get_server_msg(cmsg)

VBCS_decode_message cmsg.msgnumber, layer, origin, severity, number
```

The following table lists the CT-Library routines that a server message callback can call:

A Server Message Callback Can Call:	**Under What Circumstances?**
vbct_config_num vbct_config_str	To retrieve information only.
vbct_con_props_num vbct_con_props_str	To retrieve information or to set the CS_USERDATA property only.
vbct_cmd_props_num vbct_cmd_props_str	To retrieve information or to set the CS_USERDATA property only.
vbct_cancel (CS_CANCEL_ATTN)	Any.
vbct_do_binds, vbct_describe, vbct_fetch, vbct_data, vbct_res_info	The routines must be called with the command structure returned by the callbacks's vbct_con_props_num(CS_EED_CMD) call.

Capabilities

Capabilities describe features that a client/server connection supports.

For a list of capabilities, refer to vbct_capability.

Related Topics:

What are Capabilities Good For?

Types of Capabilities

Setting and Retrieving Capabilities

Setting and Retrieving Multiple Capabilities

What are Capabilities Good For?

An application can use capabilities to find out what features are supported by a connection's actual TDS version.

In particular, an application can:

- Find out whether a server connection supports a particular type of request.
- Tell a server not to send a particular type of response on a connection.

Types of Capabilities

There are two types of capabilities:

- CS_CAP_REQUEST capabilities, or "request capabilities," which describe the types of client requests that can be sent on a server connection.
- CS_CAP_RESPONSE capabilities, or "response capabilities," which describe the types of server responses that a connection does not wish to receive.

Setting and Retrieving Capabilities

Before calling vbct_connect to open a connection, an application can:

Retrieve request or response capabilities, to determine what request and response features are normally supported at the connection's current TDS version level.   A connection's TDS level defaults to the version level that the application requested in its call to vbct_init.   An application can change a connection's TDS level by calling vbct_con_props_num and vbct_con_props_str with property as CS_TDS_VERSION.

Set response capabilities, to indicate that a connection does not wish to receive particular types of responses.   For example, an application can set a connection's TDS_RES_NOEED capability to CS_TRUE to indicate that the connection does not wish to receive extended error data.

During the connection process, the client and server negotiate a TDS version level for the connection. The TDS version level determines which capabilities the connection will support.

After a connection is open, an application can:

- Retrieve request capabilities to find out what types of requests the connection will support.
- Retrieve response capabilities to find out whether the server has agreed to withhold the previously-indicated response types from the connection.

Setting and Retrieving Multiple Capabilities

Gateway applications often need to set or retrieve all capabilities of a type category with a single call to vbct_capability. To do this, an application calls vbct_capability with the following:

- type as the type category of interest
- capability as CS_ALL_CAPS
- value as a CS_CAP_TYPE structure

CT-Library provides the following macros to enable an application to set, clear, and test bits in a CS_CAP_TYPE structure:

CS_SET_CAPMASK(mask, capability)

CS_CLR_CAPMASK(mask, capability)

CS_TST_CAPMASK(mask, capability)

where mask is a pointer to a CS_CAP_TYPE structure and capability is the capability of interest.

CT-Library Messages

CT-Library message numbers are four bytes long.   Each byte contains a separate piece of information.

Related Topics:

What the Bytes Represent

Decoding a Message Number

Message Severities

What the Bytes Represent

The first (high-order) byte represents the CT-Library layer that is reporting the message. A typical application will not examine this byte except to provide information for SYBASE Technical Support.

The second byte represents the message's origin. A typical application will not examine this byte except to provide information for SYBASE Technical support.

The third byte represents the message's severity.

The fourth (low-order) byte represents a layer-specific message number.

Decoding a Message Number

SQL Sombrero/VBX provides a vbcs_decode_message function to help an application decode a message number:

Message Severities

The following table lists CT-Library message severities:

Severity:	Explanation:	User Action:
CS_SV_INFORM	No error has occurred. The message is informational.	No action is required.
CS_SV_CONFIG_FAIL	A SYBASE configuration error has been detected. Configuration errors include missing localization files, a missing interfaces file, and an unknown server name in the interfaces file.	Raise an error so that the application's end-user can correct the problem.
CS_SV_RETRY_FAIL	An operation has failed, but the operation can be retried. An example of this type of operation is a network read that times out.	The return value from an application's client message callback determines whether or not CT-Library retries the operation. If the client message callback returns CS_SUCCEED, CT-Library retries the operation. If the client message callback returns CS_FAIL, CT-Library does not retry the operation and marks the connection as dead. In this case, call vbct_close(CS_FORCE_CLOSE) to close the connection and then re-open it, if desired, by calling vbct_connect.
CS_SV_API_FAIL	A CT-Library routine generated an error. This error is typically caused by a bad parameter or calling sequence. The server connection is probably salvageable.	Call vbct_cancel(CS_CANCEL_ALL) to clean up the connection. If vbct_cancel(CS_CANCEL_ALL) returns CS_SUCCEED, the server connection is unharmed. Note that it is illegal to perform this type of cancel from within a client message callback routine.
CS_SV_RESOURCE_FAIL	A resource error has occurred. This error is typically caused by a malloc failure or lack of file descriptors. The server connection is probably not salvageable.	Call vbct_close (CS_FORCE_CLOSE) to close the server connection and then re-open it, if desired, by calling vbct_connect. Note that it is illegal to make these calls from within a client message callback routine.
CS_SV_COMM_FAIL	An unrecoverable error in the server communication channel has occurred. The server connection is not salvageable.	Call vbct_close (CS_FORCE_CLOSE) to close the server connection and then re-open it, if desired, by calling vbct_connect. Note that it is illegal to make these calls from within a client message callback routine.
CS_SV_INTERNAL_FAIL	An internal CT-Library error has occurred.	Call vbct_exit (CS_FORCE_EXIT) to exit CT-Library, and then exit the application. Note that it is illegal to call vbct_exit from within a client message callback routine.
CS_SV_FATAL	A serious error has occurred. All server connections are unusable.	

Call vbct_exit (CS_FORCE_EXIT) to exit CT-Library, and then exit the application. Note that it is illegal to call vbct_exit from within a client message callback routine.

Dynamic SQL

Dynamic SQL allows an application to execute SQL statements containing variables whose values are determined at run-time. It is useful for precompiler support.

An application prepares a dynamic SQL statement by associating a SQL statement containing placeholders with an identifier and sending the statement to a server to be partially compiled and stored. The statement is then known as a "prepared statement".

When an application is ready to execute a prepared statement, it defines values to substitute for the SQL statement's placeholders and sends a command to execute the statement.

Related Topics:

Preparing a Statement

Getting a Description of Prepared Statement Input

Executing Statements

Executing a Prepared Statement

Executing a Literal SQL Statement

Getting a Description of Prepared Statement Output

De-allocating a Prepared Statement

Preparing a Statement

Follow these steps to prepare a statement:

1. Call vbct_dynamic with type as CS_PREPARE to initiate a command to prepare a statement.
2. Call vbct_send to send the command to the sever.
3. Call vbct_results as necessary to proces the results of the command. A successful CS_PREPARE command will generate a CS_CMD_SUCCEED result.

Getting a Description of Prepared Statement Input

An application typically retrieves a description of prepared statement input parameters before passing input value to a prepared statement.

Follow these steps to get a description of prepared statement input parameters:

1 Call vbct_dynamic with type as CS_DESCRIBE_INPUT to initiate the command to get the description.

2. Call vbct_send to send the command to the server.

3. Call vbct_results as necessary to process the results of the command. A CS_DESCRIBE_INPUT command generates a result set of type CS_DESCRIBE_RESULT. This result set contains no fetchable data but does contain descriptive information for each of the input parameters.

An application can either call vbct_describe or call vbct_dyndesc with operation as CS_GETATT to get a description of a particular input parameter.

Executing Statements

An application can call vbct_dyamic to execute a prepared statement or a literal SQL statement.

An application typically prepares a statement if it will need to execute the statement multiple times.

An application typically executes a literal SQL statement if it will need to execute the statement only once.

Executing a Prepared Statement

Follow these steps to execute a prepared statement:

1. Call vbct_dynamic with type as CS_EXECUTE to initiate a command to execute the statement.
2. Define input parameters to the statement. An application can define input parameters to a prepared statement in two ways:
   - The applicationh can call vbct_param once for each parameter that the prepared statement requires. Most applications will use this method.
   - The application can call vbct_dyndesc to allocate a descriptor and describe the parameters.
3. Call vbct_send to send the command to the server.
4. Process the results of the command. An application can process results in two ways:
   - It can use vbct_results, vbct_do_binds and vbct_fetch to loop through results, bind result items and fetch rows.
   - It can use vbct_results, vbct_fetch and vbct_dyndesc to loop through results, fetch rows and get data item descriptions and data.

Executing a Literal SQL Statement

Follow these steps to execute a literal SQL statement:

1. Call vtct_dynamic with type as CS_EXEC_IMMEDIATE to initiate a command to execute the statement.
2. Call vbct-send to send the command to the server.
3. Call vbct_results as necessary to process the results of the command. A CS_EXEC_IMMEDIATE command does not generate fetchable results, but an application can examine vbct_results result_type parameter to determine whether the command succeeded or failed.

SQL Server imposes the following restrictions on statements executed via CS_EXEC_IMMEDIATE:

- Select statements are not allowed.
- Parameters are not allowed.

Getting a Description of Prepared Statement Output

An application typically needs to get a description of prepared statement output before processing the results of an ad hoc query.

For example, a forms-based application that processes interactive SQL queries needs to determine result column types and lengths before displaying output.

Follow these steps to get a description of prepared statement output columns:

1. Call vbct_dynamic with type as CS_DESCRIBE_OUTPUT to initiate the command to get the description.

2. Call vbct_send to send the command to the server.

3. Call vbct_results as necessary to process the results of the command. A CS_DESCRIBE_OUTPUT command generates a result set of type CS_DESCRIBE_RESULT. This result set contains no fetchable data but does contain descriptive information for each of the output columns.

An application can either call vbct_describe or call vbct_dyndesc with operation as CS_GETATT to get a description of a particular output column.

De-allocating a Prepared Statement

Follow these steps to de-allocate a prepared statement:

1. Call vbct_dynamic to initiate the command to de-allocate the statement.
2. Call vbct_send to send the command to de-allocate the statement.
3. Call vbct_results as necessary to process the results of the command.

Error and Message Handling

All CT-Library routines return success or failure indications. It is highly recommended that applications check these return codes.

In addition, CT-Library applications must handle two types of error and informational messages:

- CT-Library messages, also known as "client messages", are generated by CT-Library. They range in severity from informational messages to fatal errors.
- Server messages are generated by the server. Server messages also range in severity from informational messages to fatal errors. SQL Server messages can be listed by executing the Transact-SQL command "select from sysmessages".

NOTE: Don't confuse CT-Library and server messages with a result set of type CS_MSG_RESULT. CT-Library and server messages are the means through which CT-Library and the server communicate error and informational conditions to an application. An application accesses CT-Library and server messages either through message callback routines, or in-line, using vbct_diag. A message result set, on the other hand, is one of several types of result sets a server can return to an application. An application processes a result set of type CS_MSG_RESULT by calling vbct_res_info to get the message's id.

Related Topics:

Two Methods of Handling Messages

In-Line Message Handling

CT-Library's Message Structures

Sequencing Long Messages

Message Structure Fields for Sequenced Messages

Sequenced Messages and Extended Error Data

Sequenced Messages and vbct_diag

Extended Error Data

What's Extended Error Data Good For?

How Can an Application Tell if Extended Error Data is Available?

Server Message Callbacks and Extended Error Data

In-Line Error Handling and Extended Error Data

Server Transaction States

Retrieving Transaction States in Main-Line Code

Retrieving Transaction States in a Server Message Callback

Two Methods of Handling Messages

An application can handle CT-Library and server messages by completing one of the following:

- By installing callback routines to handle messages.
- In-line, using the CT-Library routine vbct_diag.

The callback method has the following advantages:

- Centralizing message handling code.
- Providing a method to gracefully handle unexpected errors. CT-Library automatically calls the appropriate message callback whenever a message is generated, so an application will not fail to trap unexpected errors. An application using only main-line error-handling logic may not successfully trap errors that have not been anticipated.

In-line message handling has the advantage of allowing an application to check for messages at particular times. For example, an application that is creating a connection might choose to wait until all connection-related commands are issued before checking for messages.

Most applications will use the callback method to handle messages. However, an application that is running on a platform/language combination that does not support callbacks must use the in-line method.

An application indicates which method it will use by calling vbct_install_callbacks to install message callbacks or by calling vbct_diag to initialize in-line message handling.

An application can use different methods on different connections. For example, an application can install message callbacks at the context level, allocate two connections, and then call vbct_diag to initialize in-line message handling for one of the connections. The other connection will use the default message callbacks that it picked up from its parent context.

An application can switch back and forth between the in-line and the callback methods:

- Installing either a client message callback or a server message callback turns off in-line message handling. Any saved messages are discarded.
- Likewise, calling vbct_diag to initialize in-line message handling de-installs a connection's message callbacks. If this occurs, the connection's first CS_GET call to vbct_diag will retrieve a warning message to this effect.

If a callback of the proper type is not installed and in-line message handling is not enabled, CT-Library discards message information.

Using Callbacks to Handle Messages

An application calls vbct_install_callbacks to install message callbacks.

CT-Library stores callbacks in the CS_CONNECTION and CS_CONTEXT structures. Because of this, when a CT-Library error occurs that makes a CS_CONNECTION or CS_CONTEXT structure unusable, CT-Library cannot call the client message callback. However, the routine that caused the error still returns CS_FAIL.

For more information on using callbacks to handle CT-Library and server messages, refer to Callbacks and refer to vbct_install_callbacks.

In-Line Message Handling

An application calls vbct_diag to initialize in-line message handling for a connection. A typical application calls vbct_diag immediately after calling vbct_con_alloc to allocate the connection structure.

An application cannot use vbct_diag at the context level. This means an application cannot use vbct_diag to retrieve messages generated by routines that take a CS_CONTEXT (and no CS_CONNECTION) as a parameter. These messages are unavailable to an application that is using in-line error handling.

An application that is retrieving messages into a SQLCA, SQLCODE, or SQLSTATE should set the CT-Library property CS_EXTRA_INF to CS_TRUE.

The CS_DIAG_TIMEOUT property controls whether CT-Library fails or retries when a CT-Library routine generates a timeout error.

If a CT-Library error occurs that makes a CS_CONNECTION structure unusable, vbct_diag returns CS_FAIL when called to retrieve information about the original error.

For more information on the in-line method of handling CT-Library and server messages, vbct_diag.

CT-Library's Message Structures

CT-Library uses the following structures to return message information:

- The CS_CLIENTMSG structure
- The CS_SERVERMSG structure
- The SQLCA structure
- The SQLCODE structure
- The SQLSTATE structure
- The CS_EXTRA_INF Property

The CS_EXTRA_INF property determines whether or not CT-Library returns certain kinds of informational messages.

An application that is retrieving messages into a SQLCA, SQLCODE , or SQLSTATE should set the CT-Library property CS_EXTRA_INF to CS_TRUE. This is because the SQL structures require information that CT-Library does not customarily return. If CS_EXTRA_INF is not set, a loss of information will occur.

An application that is not using the SQL structures can also set CS_EXTRA_INF to CS_TRUE. In this case, the extra information is returned as standard CT-Library messages.

The additional information returned includes the number of rows affected by the most recent command.

Sequencing Long Messages

Message callback routines and vbct_diag return CT-Library and server messages in CS_CLIENTMSG and CS_SERVERMSG structures. In the CS_CLIENTMSG structure, the message text is stored in the msgstring field. In the CS_SERVERMSG structure, the message text is stored in the text field. Both msgstring and text are CS_MAX_MSG bytes long.

If a message longer than CS_MAX_MSG - 1 bytes is generated, Client-Library's default behavior is to truncate the message. However, an application can use the CS_NO_TRUNCATE property to tell CT-Library to "sequence" long messages instead of truncating them.

When CT-Library is sequencing long messages, it uses as many CS_CLIENTMSG or CS_SERVERMSG structures as necessary to return the full text of a message. The message's first CS_MAX_MSG bytes are returned in one structure, its second CS_MAX_MSG bytes in a second structure, and so forth.

CT-Library null terminates only the last chunk of a message. If a message is exactly CS_MAX_MSG bytes long, the message is returned in two chunks; the first containing CS_MAX_MSG bytes of the message and the second containing a null terminator.

If an application is using callback routines to handle messages, CT-Library calls the callback routine once for each message chunk.

If an application is using vbct_diag to handle messages, it must call vbct_diag once for each message chunk.

NOTE: The SQLCA, SQLCODE, and SQLSTATE structures do not support sequenced messages. An application cannot use these structures to retrieve sequenced messages. Messages that are too long for these structures are truncated.

NOTE: Operating system messages, if any, are reported via the osstring field of the CS_CLIENTMSG structure. CT-Library does not sequence operating system messages.

Message Structure Fields for Sequenced Messages

The status field in the CS_CLIENTMSG and CS_SERVERMSG structures indicates whether the structure contains a whole message or a chunk of a message.

The following table lists status values that are related to sequenced messages:

Symbolic Value:	**To Indicate:**
CS_FIRST_CHUNK	The message text is the first chunk of the message.
CS_LAST_CHUNK	The message text is the last chunk of the message.

If CS_FIRST_CHUNK and CS_LAST_CHUNK are both on, then the message text in the structure is the entire message.

If neither CS_FIRST_CHUNK nor CS_LAST_CHUNK is on, then the message text in the structure is a middle chunk.

The msgstringlen field in the CS_CLIENTMSG structure and the textlen field in the CS_SERVERMSG structure always reflect the length of the current message chunk.

All other fields in the CS_CLIENTMSG and CS_SERVERMSG are repeated with each message chunk.

Sequenced Messages and Extended Error Data

If a sequenced server message has extended error data associated with it, an application can retrieve the extended error data while processing any single chunk of the sequenced message. Once the application has retrieved the extended error data, however, it is no longer available.

Sequenced Messages and vbct_diag

If an application is using sequenced error messages, vbct_diag acts on message chunks instead of messages. The following effects occur:

- A vbct_diag(CS_GET) call with index i returns the i'th message chunk, not the i'th message.
- A vbct_diag(CS_MSGLIMIT) call limits the number of chunks, not the number of messages, that CT-Library will store.
- A vbct_diag(CS_STATUS) call returns the number of currently-stored chunks, not the number of currently-stored messages.

Extended Error Data

Some server messages have "extended error data" associated with them. Extended error data is simply additional information about the error.

For SQL Server messages, the additional information is most typically which column or columns provoked the error.

CT-Library makes extended error data available to an application in the form of a parameter result set, where each result item is a piece of extended error data.   A piece of extended error data can be named, and can be of any datatype.

An application can retrieve extended error data but is not required to do so.

What's Extended Error Data Good For?

Applications that allow end-users to enter or edit data often need to report errors to their users at the column level. The standard server message mechanism, however, makes column-level information available only within the text of the server message. Extended error data provides a means for applications to conveniently access column-level information.

For example, imagine an application that allows end-users to enter and edit data in the titleauthor table in the pubs2 database. titleauthor uses a key composed of two columns, au_id and title_id. Any attempt to enter a row with an au_id and title_id that match an existing row will cause a "duplicate key" message to be sent to the application.

On receiving this message, the application needs to identify the problem column or columns to the end-user, so that the user can correct them. This information is not available in the duplicate key message, except in the message text. The information is available, however, as extended error data.

How Can an Application Tell if Extended Error Data is Available?

When CT-Library returns standard server message information to an application in a CS_SERVERMSG structure, it sets the CS_HASEED bit of the status field of the CS_SERVERMSG structure if extended error data is available for the message.

Extended error data is returned to an application in the form of a parameter result set that is available on a special CS_COMMAND structure that CT-Library provides.

To retrieve extended error data, an application processes the parameter result set.

Server Message Callbacks and Extended Error Data

Within a server message callback routine, an application retrieves the CS_COMMAND with the extended error data by calling vbct_con_props_num and vbct_con_props_str with property as CS_EED_CMD:

```
ret = vbct_con_props(connection, CS_GET, CS_EED_CMD,eed_cmd, CS_UNUSED, outlen);
```

vbct_con_props_num and vbct_con_props_str sets eed_cmd to point to the CS_COMMAND on which the extended error data is available.

Once it has the CS_COMMAND, the callback routine processes the extended error data as a normal parameter result set, calling vbct_res_info, vbct_describe, vbct_do_binds, vbct_fetch, and vbct_get_data to describe, bind, and fetch the parameters.   It is not necessary for the callback routine to call vbct_results.

In-Line Error Handling and Extended Error Data

An application that is handling server messages in-line retrieves the CS_COMMAND with the extended error data by calling vbct_diag with operation as CS_EED_CMD:

ret = vbct_diag (connection, CS_EED_CMD,CS_SERVERMSG_TYPE, index, eed_cmd);

In this call, type must be CS_SERVERMSG_TYPE and index must be the index of the message for which extended error data is available. vbct_diag sets eed_cmd to point to the CS_COMMAND on which the extended error data is available.

Once it has the CS_COMMAND, the application processes the extended error data as a normal parameter result set, calling vbct_res_info, vbct_describe, vbct_do_binds, vbct_fetch, and vbct_get_data to describe, bind, and fetch the parameters.   It is not necessary for the application to call vbct_results.

Server Transaction States

Server transaction state information is useful when an application needs to determine the outcome of a transaction.

The following table lists the symbolic values that represent transaction states:

Symbolic Value:	To Indicate:
CS_TRAN_IN_PROGRESS	A transaction is in progress.
CS_TRAN_COMPLETED	The most recent transaction completed successfully.
CS_TRAN_STMT_FAIL	The most-recently-executed statement in the current transaction failed.
CS_TRAN_FAIL	The most recent transaction failed.
CS_TRAN_UNDEFINED	A transaction state is not currently defined.

Retrieving Transaction States in Main-Line Code

In main-line code, an application retrieves a transaction state by calling vbct_res_info with type as CS_TRANS_STATE:

ret = vbct_res_info (cmd, CS_TRANS_STATE,trans_state, CS_UNUSED, outlen)

vbct_res_info sets trans_state to one of the symbolic values listed in Transaction states Table above.

Transaction state information is available only for CS_COMMAND structures with pending results or an open cursor.   This means transaction state information is available if an application's last call to vbct_results returned CS_SUCCEED.

Transaction state information is guaranteed to be correct only after vbct_results sets result_type to CS_CMD_DONE, CS_CMD_SUCCEED, or CS_CMD_FAIL.

Retrieving Transaction States in a Server Message Callback

An application can retrieve transaction states inside a server message callback only if extended error data is available.

Within a server message callback, CT-Library indicates that extended error data is available by setting the CS_HASEED bit of the status field of the CS_SERVERMSG structure describing the message.

If extended error data is available, the application can retrieve the current transaction state by:

1. Retrieving the CS_COMMAND with the extended error data by calling vbct_con_props_num and vbct_con_props_str with property as CS_EED_CMD.

2. Calling vbct_res_info with type as CS_TRANS_STATE. vbct_res_info sets its buffer parameter to one of the symbolic values listed in the Transaction states Table above.

Results

When a CT-Library command executes on a server, it can generate various types of results which are returned to the application that sent the command: The results are as follows:

- Regular row results
- Cursor row results
- Parameter results
- Stored procedure return status results
- Compute row results
- Message results
- Describe results
- Format results

Results are returned to an application in the form of "result sets."   A result set contains only a single type of result data.   Regular row and cursor row result sets can contain multiple rows of data, but other types of result sets contain at most a single row of data.

An application processes results by calling vbct_results, which indicates the type of result available by setting result_type.

vbct_results sets result_type to CS_CMD_DONE to indicate that the results of a "logical command" have been completely processed.   A logical command is generally considered to be any Open Client command defined via vbct_command, vbct_dynamic.   Exceptions to this rule are documented in "When are the Results of a Command Completely Processed?" reference to vbct_results (238 approx)

Some commands, i.e. a language command containing a Transact-SQL update statement, do not generate results.   vbct_results sets result_type to CS_CMD_SUCCEED or CS_CMD_FAIL to indicate the status of a command that does not return results.

Related Topics:

Regular Row Results

Parameter Results

Stored Procedure Return Status Results

Compute Row Results

Message Results

Describe Results

Format Results

Regular Row Results

A regular row result set is generated by the execution of a Transact-SQL select statement on a server.

A regular row result set contains zero or more rows of tabular data.

Parameter Results

A parameter result set contains a single "row" of parameters.   Several types of data can be returned as a parameter result set, including:

- Message parameters.
- Stored procedure return parameters.

Extended error data and registered procedure notification parameters are also returned as parameter result sets, but since an application does not call vbct_results to process these types of data, the application never sees a result type of CS_PARAM_RESULT.   Instead, the row of parameters is simply available to be fetched after the application retrieves the CS_COMMAND structure containing the data.

Stored Procedure Return Status Results

A status result set consists of a single row which contains a single value, a return status.

Compute Row Results

A compute row result set contains a single row of tabular data with a number of columns equal to the number of columns listed in the compute clause that generated the compute row.

Message Results

A message result set does not actually contain any data. Instead, a message has an "id." To get a message's id, an application can call vbct_res_info after vbct_results returns CS_MSG_RESULT.

If parameters are associated with a message, they are returned as a separate parameter result set, immediately following the message result set.

Describe Results

A describe result set does not contain fetchable data, but rather indicates the existence of descriptive information returned as the result of a dynamic SQL describe input or describe output command.

An application can retrieve this descriptive information by calling vbct_describe or vbct_dyndesc.

Format Results

There are two types of format results:

- regular row format results
- compute row format results

Format result sets do not contain fetchable data, but rather indicate the availability of format information for the regular row and compute row result sets with which they are associated.

All format information for a command is returned before any data. That is, the row format and compute format result sets for a command precede the regular row and compute row result sets that the command generates.

Format information is primarily of use in gateway applications, which need to repackage SQL Server results before sending them on to a foreign client.

A connection receives format results only if its CS_EXPOSE_FMTS property is set to CS_TRUE.

Types

Open Client CT-Library supports a wide range of datatypes.   These datatypes are shared with Open Client CT-Library and Open Server Server-Library.   In most cases, they correspond directly to SQL Server datatypes.

The "Datatype Summary" chart, below, lists Open Client/Server type constants, their corresponding typedefs, and their corresponding SQL Server or Secure SQL Server datatypes, if any.

A list of Open Client routines that are useful in manipulating datatypes follows the summary chart, together with more detailed information on each datatype.

Related Topics:

Datatype Summary

Routines That Manipulate Datatypes

Binary Types

Bit Types

Character Types

Datetime Types

Numeric Types

Money Types

Security Types

Text and Image Types

Open Client User-Defined Datatypes

Datatype Summary

The following table lists Open Client/Server type constants, their corresponding typedefs, and their corresponding SQL Server or Secure SQL Server datatypes, if any:

	Open Client/Server Type Constant	**Description**	**Corresponding Open Client/Server Typedef**	**Corresponding Server Datatype**
Binary types	CS_BINARY_TYPE	Binary type	CS_BINARY	binary, varbinary
	CS_LONGBINARY_TYPE	Long binary type	CS_LONGBINARY	NONE
	CS_VARBINARY_TYPE	Variable-length binary type	CS_VARBINARY	NONE
Bit types	CS_BIT_TYPE	Bit type	CS_BIT	boolean
Character types	CS_CHAR_TYPE	Character type	CS_CHAR	char, varchar
	CS_LONGCHAR_TYPE	Long character type	CS_LONGCHAR	NONE
	CS_VARCHAR_TYPE	Variable-length character type	CS_VARCHAR	NONE
Datetime types	CS_DATETIME_TYPE	8-byte datetime type	CS_DATETIME	datetime
	CS_DATETIME4_TYPE	4-byte datetime type	CS_DATETIME4	smalldatetime
Numeric types	CS_TINYINT_TYPE	1-byte integer type	CS_TINYINT	tinyint
	CS_SMALLINT_TYPE	2-byte integer type	CS_SMALLINT	smallint
	CS_INT_TYPE	4-byte integer type	CS_INT	int
	CS_DECIMAL_TYPE	Decimal type	CS_DECIMAL	decimal
	CS_NUMERIC_TYPE	Numeric type	CS_NUMERIC	numeric
	CS_FLOAT_TYPE	8-byte float type	CS_FLOAT	float
	CS_REAL_TYPE	4-byte float type	CS_REAL	real
Money types	CS_MONEY_TYPE	8-byte money type	CS_MONEY	money
	CS_MONEY4_TYPE	4-byte money type	CS_MONEY4	smallmoney
Security types	CS_BOUNDARY_TYPE	Secure SQL Server boundary type	CS_CHAR	sensitivity boundary
	CS_SENSITIVITY_TYPE	Secure SQL Server sensitivity type	CS_CHAR	sensitivity
Text and image types	CS_TEXT_TYPE	Text type	CS_TEXT	text
	CS_IMAGE_TYPE	Image type	CS_IMAGE	image

Routines That Manipulate Datatypes

Open Client CT-Library provides several routines that are useful for manipulating datatypes.   They are as follows:

- vbcs_calc, that performs arithmetic operations on decimal, money, and numeric datatypes
- vbcs_cmp, that compares datetime, decimal, money, and numeric datatypes
- vbcs_convert, that converts a data value from one datatype to another
- vbcs_dt_crack, that converts a machine readable datetime value into a user-accessible format
- vbcs_dt_info, that sets or retrieves language-specific datetime information
- vbcs_strcmp, that compares two strings

These routines are documented under Function Descriptions later on in the Manual.

Binary Types

Open Client has three binary types: CS_BINARY, CS_LONGBINARY, and CS_VARBINARY.

CS_BINARY corresponds to the SQL Server types binary and varbinary. This means CT-Library interprets both the server binary and varbinary types as CS_BINARY. For example, vbct_describe returns CS_BINARY_TYPE when describing a result column that has the server datatype varbinary.

WARNING: CS_LONGBINARY and CS_VARBINARY do not correspond to any SQL Server datatypes. Specifically, CS_VARBINARY does not correspond to the SQL Server datatype varbinary.

CS_LONGBINARY does not correspond to any SQL Server type, but some Open Server applications may support CS_LONGBINARY. An application can use the CS_DATA_LBIN capability to determine whether an Open Server connection supports CS_LONGBINARY. If it does, then vbct_describe can return CS_LONGBINARY when describing a result data item.

A CS_LONGBINARY value has a maximum length of 2,147,483,647 bytes.

CS_VARBINARY does not correspond to any SQL Server type. For this reason, Open Client routines do not return CS_VARBINARY_TYPE. CS_VARBINARY is provided to enable non-C programming language veneers to be written for Open Client. Typical client applications will not use CS_VARBINARY.

Although CS_VARBINARY variables are used to store variable-length values, CS_VARBINARY is considered to be a fixed-length type. This means that an application does not typically need to provide CT-Library with the length of a CS_VARBINARY variable. For example, vbct_do_binds ignores the value of datafmt.maxlength when binding to a CS_VARBINARY variable.

Bit Types

Open Client supports a single bit type, CS_BIT. This type is intended to hold server bit (or boolean) values of 0 or 1. When converting other types to bit, all non-zero values are converted to 1:

Character Types

Open Client has three character types, CS_CHAR, CS_LONGCHAR, and CS_VARCHAR:

CS_CHAR corresponds to the SQL Server types char and varchar. This means CT-Library interprets both the server char and varchar types as CS_CHAR. For example, vbct_describe returns CS_CHAR_TYPE when describing a result column that has the server datatype varchar.

Warning: CS_LONGCHAR and CS_VARCHAR do not correspond to any SQL Server datatypes. Specifically, CS_VARCHAR does not correspond to the SQL Server datatype varchar.

CS_LONGCHAR does not correspond to any SQL Server type, but some Open Server applications may support CS_LONGCHAR. An application can use the CS_DATA_LCHAR capability to determine whether an Open Server connection supports CS_LONGCHAR. If it does, then vbct_describe can return CS_LONGCHAR when describing a result data item.

A CS_LONGCHAR value has a maximum length of 2,147,483,647 bytes.

CS_VARCHAR does not correspond to any SQL Server type. For this reason, Open Client routines do not return CS_VARCHAR_TYPE. CS_VARCHAR is provided to enable non-C programming language veneers to be written for Open Client. Typical client applications will not use CS_VARCHAR.

Although CS_VARCHAR variables are used to store variable-length values, CS_VARCHAR is considered to be a fixed-length type. This means that an application does not typically need to provide CT-Library with the length of a CS_VARCHAR variable. For example, vbct_do_binds ignores the value of datafmt.maxlength when binding to a CS_VARCHAR variable.

Datetime Types

Open Client supports two datetime types, CS_DATETIME and CS_DATETIME4. These datatypes are intended to hold 8-byte and 4-byte datetime values, respectively.

An Open Client application can use the CT-Library Routine vbcs_dt_crack to extract date parts (year, month, day, etc.) from a datetime structure.

CS_DATETIME corresponds to the SQL Server datetime datatype. The range of legal CS_DATETIME values is from January 1, 1753 to December 31, 9999, with a precision of 1/300th of a second (3.33 milliseconds):

CS_DATETIME4 corresponds to the SQL Server smalldatetime datatype. The range of legal CS_DATETIME4 values is from January 1, 1900 to June 6, 2079, with a precision of 1 minute:

Numeric Types

Open Client supports a wide range of numeric types.

Integer types include CS_TINYINT, a 1-byte integer; CS_SMALLINT, a 2-byte integer, and CS_INT, a 4-byte integer:

CS_REAL corresponds to the SQL Server datatype real. It is implemented as a C-language float type:

CS_FLOAT corresponds to the SQL Server datatype float. It is implemented as a C-language double type:

CS_NUMERIC and CS_DECIMAL correspond to the SQL Server datatypes numeric and decimal. These types provide platform-independent support for numbers with precision and scale.

The SQL Server datatypes numeric and decimal are equivalent; and CS_DECIMAL is defined as CS_NUMERIC:

typedef CS_NUMERIC CS_DECIMAL;

where:

precision is the precision of the numeric value. At the current time, legal values for precision are from 1 to 77. The default precision is 18. CS_MIN_PREC, CS_MAX_PREC, and CS_DEF_PREC define the minimum, maximum, and default precision values, respectively.

scale is the scale of the numeric value. At the current time, legal values for scale are from 0 to 77. The default scale is 0. CS_MIN_SCALE, CS_MAX_SCALE, and CS_DEF_PREC define the minimum, maximum, and default scale values, respectively.

scale must be less than or equal to precision.

CS_DECIMAL types use the same default values for precision and scale as CS_NUMERIC types.

Money Types

Open Client supports two money types, CS_MONEY and CS_MONEY4. These datatypes are intended to hold 8-byte and 4-byte money values, respectively.

CS_MONEY corresponds to the SQL Server money datatype. The range of legal CS_MONEY values is between +/- $922,337,203,685,477.5807:

CS_MONEY4 corresponds to the SQL Server smallmoney datatype. The range of legal CS_MONEY4 values is between -$214,748.3648 and +$214,748.3647:

Security Types

Open Client supports Secure SQL Server's sensitivity_boundary and sensitivity types by defining the type constants CS_BOUNDARY_TYPE and CS_SENSITIVITY_TYPE.

These type constants differ from other Open Client type constants in that they do not correspond to similarly-named typedefs. Instead, they correspond to CS_CHAR.

This means that although Open Client routines accept and return CS_BOUNDARY_TYPE and CS_SENSITIVITY_TYPE to describe a column or variable's datatype, any corresponding program variable must be of type CS_CHAR.

For example, if an application calls vbct_do_binds with the datatype field of the CS_DATAFMT structure set to CS_SENSITIVITY_TYPE, the program variable to which the data is being bound must be of type CS_CHAR.

Text and Image Types

Open Client supports a text datatype, CS_TEXT, and an image datatype, CS_IMAGE.

CS_TEXT corresponds to the server datatype text, which describes a variable-length column containing up to 2,147,483,647 bytes of printable character data. CS_TEXT is defined as unsigned character:

CS_IMAGE corresponds to the server datatype image, which describes a variable-length column containing up to 2,147,483,647 bytes of binary data. CS_IMAGE is defined as unsigned character:

Open Client User-Defined Datatypes

An application that needs to use a datatype that is not included in the standard Open Client type set can create a user-defined datatype.

An CT-Library application creates a user-defined type by declaring it:

NOTE: Do not confuse Open Client user-defined types with SQL Server user-defined types. Open Client user-defined types are C-language types, declared within an application. SQL Server user-defined types are database column datatypes, created using the system stored procedure sp_addtype.

CompileDate

Syntax

CompileDate.

Parameters

Results

Returns a string containing the compile date of SQL Sombrero/VBX. This function will be used mainly by Technical Support to determine the version of SQL Sombrero/VBX.

Remarks

See Also

vbcs_calc

Perform an arithmetic operation on two operands.

Syntax

vbcs_calc(context&, op&, datatype&, var1(Any),var2(Any), dest(Any))

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

op&

The operation to perform. The following table lists the symbolic values that are legal for op:

Value of op:	operation:	dest is set to:
CS_ADD	Addition	var1 + var2
CS_SUB	Subtraction	var1 - var2
CS_MULT	Multiplication	var1 * var2
CS_DIV	Division	var1 / var2

datatype&

The datatype of var1, var2, and dest. The following table lists the symbolic values that are legal for datatype:

Value of datatype:	To Indicate a Datatype of:
CS_DECIMAL_TYPE	CS_DECIMAL
CS_MONEY_TYPE	CS_MONEY
CS_MONEY4_TYPE	CS_MONEY4
CS_NUMERIC_TYPE	CS_NUMERIC

var1 (Any)

A variable of the type defined by datatype&.

var2 (Any)

A variable of the type defined by datatype&.

dest (Any)

A variable of the type defined by datatype&. If vbcs_calc returns CS_FAIL, dest is not modified.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a vbcs_calc failure include:

- An invalid parameter.

- Attempted division by 0.
- Destination overflow.

Remarks

var1, var2, and dest must have the same datatype, as indicated by the datatype parameter.

In case of error, dest is not modified.

See Also

vbcs-convert

vbcs_cmp

Compare two data values.

Syntax

vbcs_cmp(context&, datatype&, var1(Any), var2(Any), result&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

datatype&

The datatype of var1 and var2. The following table lists the symbolic values that are legal for datatype:

Value of datatype:	**Indicates a Datatype of:**
CS_DATETIME_TYPE	CS_DATETIME
CS_DATETIME4_TYPE	CS_DATETIME4
CS_DECIMAL_TYPE	CS_DECIMAL
CS_MONEY_TYPE	CS_MONEY
CS_MONEY4_TYPE	CS_MONEY4
CS_NUMERIC_TYPE	CS_NUMERIC

var1 (Any)

A variable of the type defined by datatype&.

var2 (Any)

A variable of the type defined by datatype&.

result&

The following table lists the possible values for result:

Value of result:	**To Indicate:**
-1	var1 is less than var2
0	var1 is equal to var2
1	var1 is greater than var2

Results

A pointer to the result of the comparison. The following table lists the possible values for result:

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. If vbcs_cmp returns CS_FAIL, result is undefined.

The most common reason for a vbcs_cmp failure is an invalid parameter.

Remarks

vbcs_cmp sets result to indicate the result of the comparison.

var1 and var2 must have the same datatype, as indicated by the datatype parameter.

To compare string values, an application can call vbcs_strcmp.

See Also

vbcs_calc , vbcs_convert, vbcs_strcmp

vbcs_config

Set or retrieve CT-Library properties.

Syntax

vbcs_config(context&, action&, property&, buffer(Any), buflen&, outlen&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

action&

One of the following symbolic values:

Value of action:	vbcs_config:
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its default value.

property&

The property whose value is being set or retrieved. The following table lists the symbolic values that are legal for property:

Value of property:	What it is:	An application can:	buffer is:
CS_EXTRA_INF	Whether or not to return the extra information that's required when processing messages in-line, using a SQLCA, SQLCODE, or SQLSTATE.	Set, retrieve, or clear.	CS_TRUE or CS_FALSE. CS_FALSE is the default.
CS_LOC_PROP	A CS_LOCALE structure that defines localization information for this context.	Set, retrieve, or clear.	A CS_LOCALE structure previously allocated by the application.
CS_MESSAGE_CB	The CT-Library message callback routine.	Not supported in Sombrero.	If action is CS_SET, buffer is the message callback routine. If action is CS_GET, buffer is set to the address of the message callback routine that is currently installed. There is no default.
CS_USERDATA	User-allocated data.	Set, retrieve, or clear.	User-allocated data. A default is not applicable.
CS_VERSION	The version of CT-Library.	Retrieve only.	A symbol. Currently, the only legal value for CS_VERSION is CS_VERSION_100.

buffer (Any)

If a property value is being set, buffer points to the variable used in setting the property.

If a property value is being retrieved, buffer points to the variable in which vbcs_config will place the value of the property.

If a property value is being cleared, pass buffer as CS_UNUSED.

buflen&

Generally, buflen is the length, in bytes, of buffer.

If a property value is being set and the value in buffer is null-terminated, pass buflen as CS_NULLTERM.

If buffer is a fixed-length or symbolic value, pass buflen as CS_UNUSED.

outlen&

A pointer to an integer variable.

outlen is not used if a property value is being set.

If a property value is being retrieved, vbcs_config sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

If an application is setting a property value or does not care about return length information, it can pass outlen as NULL.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

See Also

vbcs_ctx_alloc , vbct_con_props , vbcs_config, vbct_init

vbcs_convert

Convert a data value from one datatype to another.

Syntax

vbcs_convert(context&, srcfmt(CS_DATAFMT), srcdata(Any),destfmt(CS_DATAFMT), destdata(Any), resultlen&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

srcfmt (CS_DATAFMT)

A CS_DATAFMT variable describing the source data format. The following table lists the fields in srcfmt that are used:

Field name:	**Set the field to:**
datatype	A type constant representing the source datatype. (CS_CHAR_TYPE, CS_BINARY_TYPE, etc.).
maxlength	The length of the data in the srcdata buffer.
locale	A pointer to a VBCS_LOCALE structure containing localization values for the source data, or NULL to use localization values from context.
All other fields	Are ignored.

srcdata (Any)

A variable containing the source data. If srcdata is NULL, the null substitution value for the datatype indicated by destfmt is placed in destdata.

CT-Library defines a default null substitution value for each datatype. An application can define custom null substitution values by calling vbcs_setnull.

destfmt (CS_DATAFMT)

A CS_DATAFMT variable describing the destination data format. The following table lists the fields in destfmt that are used:

Field name:	**Set the field to:**
datatype	A type constant representing the desired destination datatype. (CS_CHAR_TYPE, CS_BINARY_TYPE, etc.).
maxlength	The length of the destdata buffer.
locale	A pointer to a VBCS_LOCALE structure containing localization values for the destination data, or NULL to use localization values from context.
format	A bit-mask of the following symbols: For character and text destinations only: CS_FMT_NULLTERM to null-terminate the data, or CS_FMT_PADBLANK to pad to the full length of the variable with spaces. For character, binary, text, and image destinations: CS_FMT_PADNULL to pad to the full length of the variable with nulls. For any type of destination: CS_FMT_UNUSED if no format information is being provided.

scale	The scale to be used for the destination variable. If the source data is the same type as the destination, then scale can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for scale from the source data. scale must be less than or equal to precision.
precision	The precision to be used for the destination variable. If the source data is the same type as the destination, then precision can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for precision from the source data. precision must be greater than or equal to scale.
All other fields	Are ignored

destdata (Any)

A pointer to the destination buffer space.

resultlen&

A integer variable. vbcs_convert sets resultlen to the length, in bytes, of the data placed in destdata. If the conversion fails, vbcs_convert sets resultlen to CS_UNUSED.

resultlen is an optional parameter and can be passed as NULL.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

A common reason for a vbcs_convert failure is that CT-Library does not support the requested conversion.

Remarks

To determine whether a particular conversion is permitted, an application can call vbcs_will_convert.

In certain cases it can be useful to convert a datatype to itself. For instance, a conversion of CS_CHAR to CS_CHAR where srcfmt.locale and destfmt.locale define different character sets serves as a useful way to translate strings.

About Specific Conversions

A conversion to or from binary and image datatypes is a straight byte-copy, except when the conversion involves character or text. When converting character or text data to binary or image, vbcs_convert interprets the character or text string as hexadecimal, whether or not the string contains a leading "0x". When converting binary or image data to character or text, vbcs_convert creates a hexadecimal string without a leading "0x".

Converting a money, character, or text value to float can result in a loss of precision. Converting a float value to character or text can also result in a loss of precision.

Converting a float value to money can result in overflow, because the maximum CS_MONEY value is $922,337,203,685,477.5807 and the maximum CS_MONEY4 value is $214,748.3648.

If overflow occurs when converting integer or float data to character or text, the first character of the resulting value will contain an asterisk ("*") to indicate the error.

A conversion to bit has the following effect: if the value being converted is not 0, the bit value is set to 1; if the value is 0, the bit value is set to 0.

When converting decimal or numeric data to decimal or numeric data, CS_SRC_VALUE can be used in destfmt.scale and destfmt.precision to indicate that the destination data should have the same scale and precision as the source.   CS_SRC_VALUE is valid only for decimal and numeric types.

See Also

vbcs_setnull , vbcs_will_convert

vbcs_ctx_alloc

Allocate a CS_CONTEXT structure.

Syntax

vbcs_ctx_alloc(version&, context&)

Parameters

version&

The version of CT-Library behavior that the application expects. The following table lists the symbolic values that are legal for version:

Value of version:	To Indicate:	Features Supported
CS_VERSION_100	10.0 behavior	Initial version

context&

The parameter is primed with the context pointer. This context pointer is a parameter used in functions such vbcs-calc.

In case of error, vbcs_ctx_alloc sets ctx_pointer to NULL.

Results

Returns	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_MEM_ERROR	The routine failed because it could not allocate sufficient memory.
CS_FAIL	The routine failed for other reasons.

Common reasons for a vbcs_ctx_alloc failure include:

- Insufficient memory.
- Missing localization files.

Remarks

A CS_CONTEXT structure, also called a "context structure," contains information that describes an application context. For example, a context structure contains default localization information, and defines the version of CT-Library that is in use.

Allocating a context structure is the first step in any CT-Library or Server-Library application.

After allocating a CS_CONTEXT structure , a CT-Library application typically customizes the context by calling vbcs_config and/or vbct_config, and then sets up one or more connections within the context.

To de-allocate a context structure, an application can call vbcs_ctx_drop.

vbcs_ctx_global also allocates a context structure. The difference between vbcs_ctx_alloc and vbcs_ctx_global is that vbcs_ctx_alloc allocates a new context structure each time it is called, while vbcs_ctx_global allocates a new context structure only once, the first time it is called. On subsequent calls, vbcs_ctx_global simply returns a pointer to the existing context structure.

See Also

vbct_con_alloc, vbct_config_num, vbct_config_str, vbcs_ctx_global , vbcs_config

vbcs_ctx_drop

De-allocate a CS_CONTEXT structure.

Syntax

vbcs_ctx_drop(context&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

vbcs_ctx_drop returns CS_FAIL if the context contains an open connection.

Remarks

A CS_CONTEXT structure describes a particular context, or operating environment, for a set of server connections.

Once a CS_CONTEXT has been de-allocated, it cannot be re-used.   To allocate a new CS_CONTEXT, an application can call vbcs_ctx_alloc.

A CT-Library application cannot call vbcs_ctx_drop to de-allocate a CS_CONTEXT structure until it has called vbct_exit to clean up CT-Library space associated with the context.

See Also

vbcs_ctx_alloc , vbct_close, vbct_exit

vbcs_ctx_global

Allocate or return a CS_CONTEXT structure.

Syntax

vbcs_ctx_global(context&, version&)

Parameters

context&

The parameter is primed with the context pointer. This context pointer is a parameter used in functions such vbcs-calc.

version&

The version of CT-Library behavior that the application expects. The following table lists the symbolic values that are legal for version:

Value of version:	**To Indicate:**	**Features Supported**
CS_VERSION_100	10.0 behavior	Initial version

If an application has already allocated a CS_CONTEXT structure, version must match the version previously requested.

In case of error, vbcs_ctx_global sets vbctx_pointer to NULL.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_MEM_ERROR	The routine failed because it could not allocate sufficient memory.
CS_FAIL	The routine failed for other reasons.

Common reasons for a vbcs_ctx_global failure include:

- A lack of available memory.
- A version value that does not match a previously requested version.

Remarks

vbcs_ctx_alloc also allocates a context structure. The only difference between vbcs_ctx_alloc and vbcs_ctx_global is that vbcs_ctx_alloc allocates a new context structure each time it is called, while vbcs_ctx_global allocates a new context structure only once, the first time it is called. On subsequent calls, vbcs_ctx_global simply returns a pointer to the existing context structure.

Applications that need to access a single context structure from multiple independent modules use vbcs_ctx_global.

See Also

vbcs_ctx_alloc , vbcs_ctx_drop , vbcs_config , vbct_con_alloc, vbct_config_num, vbct_config_str

vbcs_decode_message

Syntax

vbcs_decode_message(msgnumber&, layer&, origin&, severity&, number&)

Parameters

msgnumber&

The CT Library message number obtained from the Client Message or Server Message event callback.

layer&

The CT Library layer that is reporting the message.

origin&

Represents the message's origin.

severity&

Represents the message's severity.   For a list of possible message severities refer to CT-Library Messages.

number&

Represents a layer-specific message number.

Results

Remarks

See Also

vbcs_get_client_msg, vbcs_get_server_msg

vbcs_diag

Manage in-line error handling.

Syntax

vbcs_diag(context&, operation&, type&, index&, buffer(Any))

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

operation&

The operation to perform. The following table lists the symbolic values that are legal for operation.

Value of operation:	vbcs_diag:	type is:	index is:	buffer is:
CS_INIT	Initializes in-line error handling.	CS_UNUSED	CS_UNUSED	NULL
CS_MSGLIMIT	Sets the maximum number of messages to store.	CS_CLIENTMSG_TYPE	CS_UNUSED	A pointer to an integer value.
CS_CLEAR	Clears message information for this context. If buffer is not NULL, vbcs_diag also clears the buffer structure by initializing it with blanks and/or NULLs, as appropriate.	One of the legal type values.	CS_UNUSED	A pointer to a structure whose type is defined by type, or NULL.
CS_GET	Retrieves a specific message.	One of the legal type values.	The one-based index of the message to retrieve.	A pointer to a structure whose type is defined by type.
CS_STATUS	Returns the current number of stored messages.	CS_CLIENTMSG_TYPE	CS_UNUSED	A pointer to an integer value.

type&

Depending on the value of operation, type indicates either the type of structure to receive message information, the type of message on which to operate, or both. The following table lists the symbolic values that are legal for type:

Value of type:	To indicate:
SQLCA_TYPE	A SQLCA structure.
SQLCODE_TYPE	A SQLCODE structure, which is a four-byte integer.
SQLSTATE_TYPE	A SQLSTATE structure, which is an array of bytes.
CS_CLIENTMSG_TYPE	A CS_CLIENTMSG structure. Also used to

indicate CT-Library messages.

index&

The index of the message of interest. The first message has an index of 1, the second an index of 2, and so forth.

buffer (Any)

A pointer to data space.

Depending on the value of operation, buffer can point to a structure or a long.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_NOMSG	The application attempted to retrieve a message whose index is higher than the highest valid index. For example, the application attempted to retrieve message number 3, when there are only 2 messages queued.

Common reasons for a vbcs_diag failure include:

- Invalid context.
- Inability to allocate memory.
- Invalid parameter combination.

Remarks

An application that includes calls to CT-Library can handle CT-Library messages in one of the following two ways:

- The application can call vbcs_install_callbacks to install a CT-Library message callback, or;
- The application can handle CT-Library messages in-line, using vbcs_diag.

It is possible for an application to switch back and forth between the in-line and the callback method:

- Installing a CT-Library message callback turns off in-line message handling. Any saved messages are discarded.

- Likewise, calling vbcs_diag to initialize in-line message handling de-installs an application's message callback. If this occurs, the application's first CS_GET call to vbcs_diag will retrieve a warning message to this effect.

If a callback of the proper type is not installed and in-line message handling is not enabled, CT-Library discards message information.

vbcs_diag manages in-line message handling for a specific context. If an application has more than one context, it must make separate vbcs_diag calls for each context.

vbcs_diag allows an application to retrieve message information into a CS_CLIENTMSG structure or a SQLCA, SQLCODE, or SQLSTATE structure. When retrieving messages, vbcs_diag assumes that buffer points to a structure of the type indicated by type.

An application that is retrieving messages into a SQLCA, SQLCODE, or SQLSTATE structure must set the CT-Library context property CS_EXTRA_INF to CS_TRUE. This is because the SQL structures contain information that is not ordinarily returned by CT-Library's error handling mechanism.

An application that is not using the SQL structures can also set CS_EXTRA_INF to CS_TRUE. In this case, the extra information is returned as standard CT-Library messages.

If vbcs_diag does not have sufficient internal storage space in which to save a new message, it throws away all unread messages and stops saving messages. The next time it is called with operation as CS_GET, it returns a special message to indicate the space problem:

After returning this message, vbcs_diag starts saving messages again.

Initializing In-Line Error Handling

To initialize in-line error handling, an application calls vbcs_diag with operation as CS_INIT.

Generally, if a context will use in-line error handling, an application should call vbcs_diag to initialize in-line error handling for the context immediately after allocating it.

Clearing Messages

To clear message information for a context, an application calls vbcs_diag with operation as CS_CLEAR.

- vbcs_diag assumes that buffer points to a structure of type type.
- vbcs_diag clears the buffer variable by setting it to blanks and/or NULLs, as appropriate.

Message information is not cleared until an application explicitly calls vbcs_diag with operations as CS_CLEAR. Retrieving a message does not remove it from the message queue.

Retrieving Messages

To retrieve message information, an application calls vbcs_diag with operation as CS_GET, type as the type of structure in which to retrieve the message, index as the one-based index of the message of interest, and buffer as a variable of the appropriate type.

vbcs_diag fills in the buffer variable with the message information.

If an application attempts to retrieve a message whose index is higher than the highest valid index, vbcs_diag returns CS_NOMSG to indicate that no message is available.

Limiting Messages

Applications running on platforms with limited memory may want to limit the number of messages that CT-Library saves.

To limit the number of saved messages, an application calls vbcs_diag with operation as CS_MSGLIMIT and type as CS_CLIENTMSG_TYPE.

When a message limit is reached, CT-Library discards any new messages.

An application cannot set a message limit that is less than the number of messages currently saved.

CT-Library's default behavior is to save an unlimited number of messages. An application can restore this default behavior by setting a message limit of CS_NO_LIMIT.

Retrieving the Number of Messages

To retrieve the number of current messages, an application calls vbcs_diag with operation as CS_STATUS and type as the CS_CLIENTMSG_TYPE.

See Also

vbct_install_callbacks, vbct_options_num, vbct_options_str

vbcs_dt_crack

Convert a machine-readable datetime value into a user-accessible format.

Syntax

vbcs_dt_crack(context&, datetype&, dateval(Any), daterec(CS_DATEREC))

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

datetype&

The type of the datetime value.   The following table lists the symbolic values that are legal for datetype:

Value of datetype:	**To Indicate:**
CS_DATETIME_TYPE	A CS_DATETIME dateval.
CS_DATETIME4_TYPE	A CS_DATETIME4 dateval.

dateval (Any)

A pointer to the datetime value to be converted.

daterec (CS_DATEREC)

A CS_DATEREC variable.   vbcs_dt_crack fills this structure with the translated datetime value.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a vbcs_dt_crack failure is an invalid parameter.

Remarks

vbcs_dt_crack converts a datetime value into its integer components and places those components into a CS_DATEREC variable.

Datetime values are stored in an internal format.   For example, a CS_DATETIME value is stored as the number of days since January 1, 1900 plus the number of 300th's of a second since midnight. vbcs_dt_crack converts a value of this type into a format that an application can more easily access.

See Also

vbcs_dt_info

vbcs_dt_info

Set or retrieve language-specific datetime information.

Syntax

vbcs_dt_info(context&, action&, locale&, type&, item&, buffer(Any), buflen&, outlen&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

action&

One of the following symbolic values:

Value of action:	vbcs_dt_info:
CS_SET	Sets a datetime conversion format.
CS_GET	Retrieves datetime information.

locale&

A CS_LOCALE structure contains locale information, including datetime information.

When setting datetime information, locale must be supplied.

When retrieving datetime information, locale can be NULL.   If locale is NULL, vbcs_dt_info uses the default locale information contained in context.

type&

The type of information of interest.   The following table lists the symbolic values that are legal for type.

Value of type:	vbcs_dt_info:	action can be:	item can be:	buffer is:
CS_MONTH	Retrieves the month name string.	CS_GET	0 - 11	A character string.
CS_SHORTMONTH	Retrieves the short month name string.	CS_GET	0 - 11	A character string.
CS_DAYNAME	Retrieves the day name string.	CS_GET	0 - 6	A character string.
CS_DATEORDER	Retrieves the date order string.	CS_GET	CS_UNUSED	A string containing the three characters "m", "d", and "y," to indicate the position of the month, day, and year in the national language's datetime format.
CS_12HOUR	Retrieves whether or not the language uses 12-hour time formats.	CS_GET	CS_UNUSED	CS_TRUE if 12-hour formats are used; CS_FALSE if 24-hour formats are used.
CS_DT_CONVFMT	Sets or retrieves the datetime conversion format.	CS_GET or CS_SET	CS_UNUSED	A symbolic value. See the Results section, below,for a

list of possible values.

item&

When retrieving information, item is the item number of the type category to retrieve. For example, to retrieve the name of the first month, an application passes type as CS_MONTH and item as 0.

When setting a datetime conversion format, pass item as CS_UNUSED.

buffer (Any)

If datetime information is being retrieved, buffer is the variable into which vbcs_dt_info will place the requested information.

If buflen indicates that buffer is not large enough to hold the requested information, vbcs_dt_info sets outlen to the length of the requested information and returns CS_FAIL.

If a datetime conversion format is being set, buffer points to a symbolic value representing a conversion format.

buflen&

The length, in bytes, of buffer.

If item is CS_12HOUR, pass buflen as CS_UNUSED.

outlen&

A pointer to an integer variable.

vbcs_dt_info sets outlen to the length, in bytes, of the requested information.

If the requested information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a vbcs_dt_info failure is an invalid parameter.

Remarks

vbcs_dt_info sets or retrieves national language-specific datetime information:

vbcs_dt_info can return national language date part names, date part ordering information, datetime format information, and whether or not the language uses 12-hour date formats.

vbcs_dt_info can set datetime format information.

vbcs_dt_info looks for national language locale information in context. An application can set locale information for a CS_CONTEXT by calling vbcs_config with property as CS_LOC_PROP.

If not specifically set, locale information in a CS_CONTEXT defaults to information that CT-Library picks up from the operating system when the context is allocated. If locale information is not available from the operating system, CT-Library uses platform-specific localization values in the new context.

The following table lists the values that are legal for buffer when type is CS_DT_CONVFMT:

Symbolic Value:	To Indicate:
CS_DATES_HMS	hh:mm:ss 16:15:31
CS_DATES_SHORT	monthname dd yyyy hh:mm [am|pm] October 19 1961 04:15:31 pm
CS_DATES_LONG	monthname dd yyyy hh:mm:ss:zzz [am|pm] October 19 1961 04:15:31:665 pm
CS_DATES_MDY1	mm/dd/yy 10/19/61
CS_DATES_MYD1	mm/yy/dd 10/61/19
CS_DATES_DMY1	dd/mm/yy 19/10/61
CS_DATES_DYM1	dd/yy/mm 19/61/10
CS_DATES_YDM1	yy/dd/mm 61/19/10
CS_DATES_YMD2	yy/mm/dd 61/10/19
CS_DATES_MDY1_YYYY	mm/dd/yyyy 10/19/1961
CS_DATES_DMY1_YYYY	dd/mm/yyyy 19/10/1961
CS_DATES_YMD2_YYYY	yyyy/mm/dd 1961/10/19
CS_DATES_DMY2	dd.mm.yy 19.10.61
CS_DATES_YMD1	yy.mm.dd 61.10.19
CS_DATES_DMY2_YYYY	dd.mm.yyyy 19.10.1961
CS_DATES_YMD1_YYYY	yyyy.mm.dd 1961.10.19
CS_DATES_DMY4	dd monthname yy 19 January 61
CS_DATES_DMY4_YYYY	dd monthname yyyy 19 January 1961
CS_DATES_MDY2	monthname dd, yy January 19,61
CS_DATES_MDY2_YYYY	monthname dd, yyyy January 19,1961
CS_DATES_DMY3	dd-mm-yy 19-10-61
CS_DATES_MDY3	mm-dd-yy 10-19-61
CS_DATES_DMY3_YYYY	dd-mm-yyyy 19-10-1961
CS_DATES_MDY3_YYYY	mm-dd-yyyy 10-19-1961
CS_DATES_YMD3	yymmdd 611019
CS_DATES_YMD3_YYYY	yyyymmdd 19611019

See Also

vbcs_dt_crack

vbcs_get_client_msg

Syntax

vbcs_get_client_msg(cmsg(CS_SERVERMSG))

Parameters

cmsg(CS_CLIENTMSG)

The parameter is a CS_CLIENTMSG variable which will be filled with the CT Library Client Message information.

See Also

vbcs_get_server_msg

vbcs_get_server_msg

Syntax

vbcs_get_server_msg(cmsg(CS_SERVERMSG))

Parameters

cmsg(CS_SERVERMSG)

The parameter is a CS_SERVERMSG variable which will be filled with the CT Library Server Message information.

See Also

vbcs_get_client_msg

vbcs_loc_alloc

Allocate a CS_LOCALE structure.

Syntax

vbcs_loc_alloc(context&, loc_pointer&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

loc_pointer&

The parameter is primed with the locale pointer. This locale pointer is a parameter used in functions such as vbcs_locale.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a vbcs_loc_alloc failure is a lack of adequate memory.

Remarks

An Open Client/Server application can use a CS_LOCALE structure to define custom localization values for a context, thread, connection, or data element. To define custom localization values, an application:

- Calls vbcs_loc_alloc to allocate a CS_LOCALE structure.
- Calls vbcs_locale(CS_SET) to load the CS_LOCALE with custom values.
- Uses the CS_LOCALE to set the CS_LOC_PROP property for a context or connection; calls srv_thread_props to set the SRV_T_LOCALE property for a thread; uses the CS_LOCALE in a CS_DATAFMT structure that describes a program variable; or uses the CS_LOCALE as a parameter to an Open Client/Server routine.
- Calls vbcs_loc_drop to drop the CS_LOCALE.

Localization values define the following:

- The language and character set to use for Open Client/Server and SQL Server messages
- How to represent dates and times
- The character set to use when converting data to and from character datatypes
- The collating sequence used to define the sort order used by vbcs_strcmp

See Also

vbcs_ctx_alloc , vbcs_loc_drop , vbcs_locale

vbcs_loc_drop

De-allocate a CS_LOCALE structure.

Syntax

vbcs_loc_drop(context&, locale&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

locale&

A pointer to a CS_LOCALE structure.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Remarks

A CS_LOCALE structure contains localization information.

Once a CS_LOCALE structure has been de-allocated, it cannot be reused.   To allocate a new CS_LOCALE structure, an application can call vbcs_loc_alloc.

An application should take care to ensure that it does not de-allocate a CS_LOCALE structure that is still in use.   A CS_LOCALE   structure is considered to be in use if a CS_DATAFMT structure references it.

An application can de-allocate a CS_LOCALE structure after calling vbcs_config or vbct_con_props_num and vbct_con_props_str to set the CS_LOC_PROP property for a context or connection.   This is because vbcs_config and vbct_con_props_num and vbct_con_props_str copy information from the user-supplied CS_LOCALE structure rather than setting up direct references to it.

See Also

vbcs_loc_alloc , vbcs_locale

vbcs_locale

Load a CS_LOCALE structure with localization values or retrieve the locale name previously used to load a CS_LOCALE structure.

Syntax

vbcs_locale(context&, action&, locale&, type&, buffer$, outlen&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

action&

One of the following symbolic values:

Value of action:	**vbcs_locale:**
CS_SET	Loads the VBCS_LOCALE with new localization values.
CS_GET	Retrieves the locale name that was used to load the VBCS_LOCALE.

locale&

A pointer to a CS_LOCALE structure obtained from vbcs_loc_alloc. If action is CS_SET, vbcs_locale modifies this structure. If action is CS_GET, vbcs_locale examines the structure to determine the locale name that was previously used to load it.

type&

The type of localization information of interest. The following table lists the symbolic values that are legal for type:

Value of type:	**To Indicate:**
CS_LC_ALL	All types of localization information. CS_LC_ALL is "set only"; that is, action must be CS_SET when type is CS_LC_ALL.
CS_LC_COLLATE	The collating sequence (also called "sort order"). Open Client uses a collating sequence when sorting and comparing character data.
CS_LC_CTYPE	The character set. Open Client uses a character set when it converts to or from character datatypes.
CS_LC_MESSAGE	The language and character set to use for Open Client/Server and SQL Server error messages.
CS_LC_TIME	The language and character set to use when converting between datetime and character datatypes. CS_LC_TIME controls month names and abbreviations, datepart ordering, and whether the "am/pm" string is used.
CS_SYB_LANG, CS_SYB_CHARSET, CS_SYB_SORTORDER, CS_SYB_LANG_CHARSET	For information on these values, see "Using Language, Character Set, and Sort Order Names With vbcs_locale. Refer to vbcs_locale.

WARNING: Open Server application programmers must set type to CS_LC_ALL when configuring the

VBCS_LOCALE structure that applies to the Open Server application as a whole.

buffer$

If action is CS_SET, buffer points to a character string that represents a locale name.

This locale name must correspond to an entry in the SYBASE locales file.

If buffer is NULL, vbcs_locale searches the operating system for a locale name to use. If an appropriate locale name cannot be found in the operating system environment, vbcs_locale uses a platform-dependent default locale name.

If action is CS_GET, buffer points to the space in which vbcs_locale will place a locale name.

outlen&

A pointer to an integer variable.

outlen is not used if action is CS_SET.

NOTE: The length is determined by the string that is passed.

If action is CS_GET and outlen is supplied, vbcs_locale sets outlen to the length, in bytes, of the locale name.

If the name is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the name.

If action is CS_SET or an application or does not care about return length information, it can pass outlen as NULL.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a vbcs_locale failure include:

- action is CS_SET and the buffer locale name cannot be found in the SYBASE locales file.
- action is CS_GET and buflen indicates that the buffer data space is too small.
- Missing localization files.

Remarks

vbcs_locale(CS_SET) loads a VBCS_LOCALE structure with localization values. vbcs_locale(CS_GET) retrieves the locale name that was previously used to load the VBCS_LOCALE.

A locale name is a character string that represents a language/character set/sort order triple. For example, the locale name "fr" might represent the language/character set/sort order triple "French, iso_1, binary". SYBASE pre-defines some locale names, which are listed in the locales file, but a system administrator can define additional locale names and add them to this file.

Loading a VBCS_LOCALE structure

An application needs to load a VBCS_LOCALE before using it to define custom localization values for a context, connection, or data element.

After loading a VBCS_LOCALE with custom values, an application can:

- Call vbcs_config with property as CS_LOC_PROP to copy the custom localization values into a context structure.
- Call vbct_con_props_num and vbct_con_props_str with property as CS_LOC_PROP to copy the custom localization values into a connection structure.
- Supply the VBCS_LOCALE as a parameter to a routine that accepts custom localization values (vbcs_convert, vbcs_strcmp, vbcs_time).

Because vbcs_config copies locale information, an application can de-allocate a VBCS_LOCALE structure after calling vbcs_config to set the CS_LOC_PROP property. Likewise, an application can de-allocate a VBCS_LOCALE structure after calling vbct_con_props_num and vbct_con_props_str to set the CS_LOC_PROP property. If a CS_DATAFMT structure uses a VBCS_LOCALE structure, however, the application must not de-allocate the VBCS_LOCALE until the CS_DATAFMT no longer references it.

The first time a locale name is referenced, all localization information for the language, character set, and sort order that the locale name identifies is read from the environment and cached into context. If this locale name is referenced again, vbcs_locale will read the information from the CS_CONTEXT instead of the environment.

Retrieving a Locale Name

An application can retrieve the locale name that was used to load a VBCS_LOCALE by calling vbcs_locale(CS_GET) with type as the type of localization information of interest and locale as a pointer to the VBCS_LOCALE structure.

vbcs_locale sets buffer to a null-terminated character string representing the locale name that was used to load the VBCS_LOCALE.

Using Language, Character Set, and Sort Order Names With vbcs_locale

It is possible for an application to use language, character set, and sort order names, instead of a locale name, when calling vbcs_locale.

To do this, an application calls vbcs_locale with type as CS_SYB_LANG, CS_SYB_CHARSET, CS_SYB_SORTORDER, or CS_SYB_LANG_CHARSET.

The following table summarizes vbcs_locale's parameters for these values of type:

Value of type:	**action is:**	**buffer is:**	**vbcs_locale:**
CS_SYB_LANG	CS_SET	A pointer to a language name.	Loads the VBCS_LOCALE with the specified language information.
	CS_GET	A pointer to data space.	Places the current language name in buffer.
CS_SYB_CHARSET	CS_SET	A pointer to a character set name.	Loads the VBCS_LOCALE with the specified character set information.
	CS_GET	A pointer to data space.	Places the current character set name in buffer.
CS_SYB_SORTORDER	CS_SET	A pointer to a sort order name.	Loads the VBCS_LOCALE with the specified sort order information.

	CS_GET	A pointer to data space.	Places the current sort order name in buffer.
CS_SYB_LANG_CHARSET	CS_SET	A pointer to a string of the form language_name. character_set_name.	Loads the VBCS_LOCALE with the specified language and character set information.
	CS_GET	A pointer to data space.	Places a string of the form language_name. character_set_name in buffer.

The application must have previously loaded the VBCS_LOCALE structure with consistent information by calling vbcs_locale with type as CS_LC_ALL.

If an application specifies only a language name, then vbcs_locale uses the character set and sort order already specified in the pre-loaded VBCS_LOCALE structure.

If an application specifies only a character set name, then vbcs_locale uses the language and sort order already specified in the pre-loaded VBCS_LOCALE structure.

If an application specifies only a sort order name, then vbcs_locale uses the language and character set already specified in the pre-loaded VBCS_LOCALE structure.

If a language/character set/sort order combination is not valid, vbcs_locale returns CS_FAIL.

If the required localization files for the requested language or character set do not exist, vbcs_locale returns CS_FAIL.

See Also

vbcs_loc_alloc , vbcs_loc_drop

vbcs_objects

Save, retrieve, or clear objects and data associated with them.

Syntax

vbcs_objects(context&, action&, objname(CS_OBJNAME),objdata(CS_OBJDATA))

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

action&

One of the following symbolic values:

Value of action:	vbcs_objects:
CS_SET	Saves an object.
CS_GET	Retrieves the first matching object that it finds.
CS_CLEAR	Clears all matching objects

objname (CS_OBJNAME)

A pointer to an object name structure. objname names and describes the object of interest.

The following table describes the CS_OBJNAME fields:

Field:	Description:	Notes:
thinkexists	Indicates whether the application expects this object to exist.	The value of thinkexists affects vbcs_objects' return code. For more information, refer to the Results section.
object_type	The type of the object.	This field is the first part of a 5-part key. Legal values are CS_CONNECTNAME CS_STATEMENTNAME CS_CURRENT_CONNECTION CS_WILDCARD (matches any value) A user-defined value. User-defined values must be >=100.
last_name	The "last name" associated with the object of interest, if any.	This field is the second part of a 5-part key.
lnlen	The length, in bytes, of last_name.	Can be CS_NULLTERM to indicate a null-terminated last_name. Can be CS_UNUSED to indicate an internal "unused" value for last_name. For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match with any last_name value.
first_name	The "first name" associated with the object of interest, if any.	This field is the third part of a 5-part key.
fnlen	The length, in bytes, of first_name.	Can be CS_NULLTERM to indicate a null-terminated first_name. Can be CS_UNUSED to indicate an internal "unused" value for first_name. For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match with

		any first_name value.
scope	Data that describes the scope of the object.	This field is the fourth part of a 5-part key.
scopelen	The length, in bytes, of scope.	Can be CS_NULLTERM to indicate null-terminated scope data. Can be CS_UNUSED to indicate an internal "unused" value for scope. For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match with any scope value.
thread	Platform-specific data that is used to distinguish threads in a multi-threaded execution environment.	This field is the fifth part of a 5-part key.
threadlen	The length, in bytes, of thread.	Can be CS_NULLTERM to indicate null-terminated thread data. Can be CS_UNUSED to indicate an internal "unused" value for thread. For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match with any thread value.

objdata (CS_OBJDATA)

A pointer to a CS_OBJDATA variable. objdata is the object of interest and any data associated with it.

The following table describes the CS_OBJDATA fields:

Value of action:	**objname is:**	**objdata is:**
CS_SET	A five-part key for the object.	The object to save, and any additional data to save with it.
CS_GET	A five-part key for the object.	Set to the retrieved object.
CS_CLEAR	A five-part key for the object.	CS_UNUSED

Results

vbcs_objects Called With		**vbcs_objects Returns**		
Action&	**objname.thinkexists**	**No Match:**	**Last-Name Match:**	**Full Match:**
CS_GET	CS_TRUE	CS_FAIL	CS_FAIL	CS_SUCCEED
CS_GET	CS_FALSE	CS_SUCCEED	CS_SUCCEED	CS_SUCCEED
CS_SET	CS_TRUE	CS_FAIL	CS_FAIL	CS_SUCCEED
CS_SET	CS_FALSE	CS_SUCCEED	CS_SUCCEED	CS_FAIL
CS_CLEAR	CS_TRUE	CS_FAIL	CS_FAIL	CS_SUCCEED
CS_CLEAR	CS_FALSE	CS_SUCCEED	CS_SUCCEED	CS_SUCCEED

Remarks

vbcs_objects is useful in precompiler applications that need to retrieve structures and data items by name.

vbcs_objects uses a five-part key, composed of the object_type, last_name, first_name, scope, and thread fields of objname structure.

- On CS_SET operations, vbcs_objects uses this key to store the objdata object.

- On CS_GET operations, vbcs_objects uses this key to retrieve an object specification into objdata.
- On CS_CLEAR operations, vbcs_objects clears all objects that match the key.

The following table describes the rules that vbcs_objects uses to determine whether or not key fields match:

objname Key Value:	**Stored Key Value:**	
	CS_UNUSED	**Other Legal Value**
CS_WILDCARD	Match	Match
CS_UNUSED	Match	No Match
Other Legal Value	No Match	Match if the values are equal and of the same length.

vbcs_objects can achieve two types of matches:

1. "last-name matches," in which the last_name, scope, and thread parts of the key match.
2. "full matches," in which all five parts of the key match.

The type of match that vbcs_objects achieves, together with action and objname.thinkexists, determine its return code.

On CS_GET and CS_CLEAR operations, an application can specify CS_WILDCARD for one or more objname key fields:

- On a CS_GET operation, vbcs_objects sets objdata to reflect the first matching object that it finds.
- On a CS_CLEAR operation, vbcs_objects clears all matching objects.

If an application has previously saved a CS_CURRENT_CONNECTION object, it can retrieve the current connection by:

- Calling vbcs_objects with objname.object_type as CS_CURRENT_CONNECTION, lnlen as CS_UNUSED, and fnlen as CS_UNUSED. vbcs_objects ignores the last_name and first_name fields of objname, and sets objdata.buffer to the name of the current connection and objdata.buflen to the length of this name.
- Calling vbcs_objects with objname.object_type as CS_CONNECTNAME and objname.last_name and objname.lnlen as the newly-retrieved connection name and name length. vbcs_objects sets objdata to the retrieved connection.

vbcs_objects(CS_SET) is not asynchronous-safe. Because of this, an application cannot call vbcs_objects(CS_SET) from within a completion callback routine.

See Also

vbcs_ctx_alloc

vbcs_setnull

Define a null substitution value to be used when binding or converting NULL data.

Syntax

vbcs_setnull(context&, datafmt(CS_DATAFMT), buffer(Any), buflen&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

datafmt (CS_DATAFMT)

A pointer to a CS_DATAFMT structure describing the datatype for which a null substitution value is being defined.

buffer (Any)

A pointer to the null substitution value. buffer's datatype must match datafmt.type.

buflen&

The length, in bytes, of buffer.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a vbcs_setnull failure include:

- A memory allocation error
- An invalid parameter

Remarks

If ANSI -style binds are in effect, CT-Library does not use null substitution values. To activate ANSI-style binds, an application sets the CT-Library property CS_ANSI_BINDS to CS_TRUE.

When ANSI-style binds are not in effect and source data for a conversion is NULL, CT-Library sets the destination data to the predefined null substitution value for that destination type. For example, converting a NULL value of any type to a CS_CHAR destination results in an empty string.

In a CT-Library application, null substitution values are defined at the context level. When a CT-Library connection is allocated, it picks up null substitution values from its parent context.

When converting a NULL source value to a CS_CHAR or CS_BINARY destination variable, CT-Library first puts 0 bytes into the destination and then uses the format field of the CS_DATAFMT structure that describes the destination to determine whether to pad or null-terminate.

To reinstate CT-Library's original default null substitution value for a particular datatype, an application can call vbcs_setnull with buffer as NULL.

CT-Library and CT-Library use the following default null substitution values:

Destination Type	Null Substitution Value
CS_BINARY_TYPE	Empty array
CS_VARBINARY_TYPE	Empty array
CS_BIT_TYPE	0
CS_CHAR_TYPE	Empty string
CS_VARCHAR_TYPE	Empty string
CS_DATETIME_TYPE	8 bytes of zeros
CS_DATETIME4_TYPE	4 bytes of zeros
CS_TINYINT_TYPE	0
CS_SMALLINT_TYPE	0
CS_INT_TYPE	0
CS_DECIMAL_TYPE	0.0 (with default scale and precision)
CS_NUMERIC_TYPE	0.0 (with default scale and precision)
CS_FLOAT_TYPE	0.0
CS_REAL_TYPE	0.0
CS_MONEY_TYPE	$0.0
CS_MONEY4_TYPE	$0.0
CS_BOUNDARY_TYPE	Empty string
CS_SENSITIVITY_TYPE	Empty string
CS_TEXT_TYPE	Empty string
CS_IMAGE_TYPE	Empty array

See Also

vbcs_will_convert

vbcs_strcmp

Compare two strings using a specified sort order.

Syntax

vbcs_strcmp(context&, locale&, type& str1$, str2$, result&)

Parameters

context&

The parameter passed is the context pointer which is passed from the vbcs_ctx_alloc function.

locale&

The parameter passed is the locale pointer that is obtained from the vbcs_loc_alloc.

An application can call vbcs_locale with type as CS_LC_COLLATE or CS_SYB_SORTORDER to change the collating sequence in a VBCS_LOCALE structure.

locale can be NULL. If locale is NULL, vbcs_strcmp uses whatever localization information is defined in the context CS_CONTEXT structure. Localization information is always defined at the context level, because a CS_CONTEXT picks up default localization information when it is allocated.

type&

The type of comparison to perform.

If type is CS_COMPARE, vbcs_strcmp performs a lexicographic comparison.

If type is CS_SORT, the values are compared as they would appear in a sorted list. It is possible for strings that are lexicographically equal to belong in different places in a sorted list.

str1$

A pointer to the first string for the comparison.

str2$

A pointer to the second string for the comparison.

result&

A pointer to the result of the comparison. The following table lists the possible values for result:

Value of result:	**To Indicate:**
<0	str1 is lexicographically less than str2, or str1 appears before str2 in a sorted list.
0	str1 is lexicographically equal to str1, or str1 is identical to str2.
>0	str1 is lexicographically greater than str2, or str1 appears after str2 in a sorted list.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.

CS_FAIL	The routine failed.

Remarks

vbcs_strcmp sets result to indicate the result of the comparison.

Some languages contain strings that are lexicographically equal according to a specific sort order, but contain different characters. Even though they are "equal," there is a standard order used when placing them into a sorted list.

For example, given a case-insensitive sort order that specifies that uppercase characters appear before lowercase characters in a sorted list, the strings "ABC" and "abc" are lexicographically equal, but do not appear in the same place in a sorted list.

An application can use vbcs_strcmp to compare strings either lexicographically or with respect to how they appear in a sorted list.

vbcs_strcmp determines which sort order to use by examining locale, (or context, if locale is NULL).

- To change the sort order in a VBCS_LOCALE structure, an application calls vbcs_locale with type as CS_LC_COLLATE or CS_SYB_SORTORDER.

- To change the sort order in a CS_CONTEXT structure, an application must first set up a VBCS_LOCALE structure with the desired sort order and then call vbcs_config to set the CS_LOC_PROP property for the context.

See Also

vbcs_cmp ,vbcs_locale , vbcs_config

vbcs_time

Retrieve the current date and time.

Syntax

vbcs_time(context&, locale&, buffer$, buflen&, outlen&, daterec(CS_DATEREC))

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

locale&

A pointer to a CS_LOCALE structure. A CS_LOCALE structure contains locale information, including formatting information that vbcs_time uses to create a current datetime string.

locale can be NULL. If locale is NULL, vbcs_time uses whatever localization information is defined in the context CS_CONTEXT structure. Localization information is always defined at the context level, because a CS_CONTEXT picks up default localization information when it is allocated.

buffer$

A pointer to the space in which vbcs_time will place a character string representing the current date and time.

buffer is an optional parameter and can be passed as NULL. If buffer is NULL, daterec must be supplied.

buflen&

The length, in bytes, of buffer.

If buffer is supplied and buflen indicates that buffer is not large enough to hold the current datetime string, vbcs_time sets outlen to the length of the datetime string and returns CS_FAIL.

If buffer is NULL, pass buflen as CS_UNUSED.

outlen&

A pointer to an integer variable.

vbcs_time sets outlen to the length, in bytes, of the current datetime string.

If the string is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the string.

If buffer is NULL, pass outlen as NULL.

If an application does not care about return length information, it can pass outlen as NULL.

daterec (CS_DATEREC)

A pointer to a CS_DATEREC structure in which vbcs_time will place the current date and time. Note that vbcs_time does not fill in the datemsecond and datetzone fields of the CS_DATEREC structure.

For more information on the CS_DATEREC structure, refer to vbcs_dt_crack.

daterec is an optional parameter and can be passed as NULL. If daterec is NULL, buffer must be supplied.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a vbcs_time failure include:

- An invalid parameter.
- buflen indicates that the buffer data space is not large enough to hold the formatted datetime string.

Remarks

vbcs_time returns the current date and time either in character string format or in a CS_DATEREC structure, or both.

vbcs_time formats the date and time according to locale information contained in context.

See Also

vbcs_dt_crack , vbcs_dt_info , vbcs_locale

vbcs_will_convert

Indicate whether a specific datatype conversion is available in the CT/Server libraries.

Syntax

vbcs_will_convert(context&, srctype&, desttype&, result&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

srctype&

The datatype of the source data.

desttype&

The datatype of the destination data.

result&

A pointer to a boolean variable.vbcs_will_convert sets result to CS_TRUE if the datatype conversion is supported and CS_FALSE if the datatype conversion is not supported.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Remarks

vbcs_will_convert allows an application to determine whether vbcs_convert is capable of performing a specific conversion.   When vbcs_convert is asked to perform a conversion that it doesn't support, it returns CS_FAIL and generates a CT-Library error.

See Also

vbcs_convert, vbcs_setnull

vbct_cancel

Cancel a command or the results of a command.

Syntax

vbct_cancel(connection&, cmd&, type&)

Parameters

connection&

The parameter passed is the connection pointer which is passed from the vbct_con_alloc function.

For CS_CANCEL_CURRENT cancels, connection must be NULL.

For CS_CANCEL_ATTN and CS_CANCEL_ALL cancels, one of connection or cmd must be NULL. If connection is supplied and cmd is NULL, the cancel operation applies to all commands pending for this connection.

cmd&

A pointer to the CS_COMMAND structure managing a client/ server operation.

For CS_CANCEL_CURRENT cancels, cmd must be supplied.   The cancel operation applies only to the results pending for this command structure.

For CS_CANCEL_ATTN and CS_CANCEL_ALL cancels, if cmd is supplied and connection is NULL, the cancel operation applies only to the command pending for this command structure.   If cmd is NULL and connection is supplied, the cancel operation applies to all commands pending for this connection.

type&

The type of cancel.   The following table lists the symbolic values that are legal for type

Value of type:	**Behavior of vbct_cancel:**	**Notes:**
CS_CANCEL_ALL	vbct_cancel sends an attention to the server, instructing it to cancel the current command. CT-Library immediately discards all results generated by the command.	Causes this connection's cursors to enter an undefined state. To determine the state of a cursor, an application can call vbct_cmd_props_num, and vbct_cmd_props_str with property as CS_CUR_STATUS.
CS_CANCEL_ATTN	vbct_cancel sends an attention to the server, instructing it to cancel the current command. The next time the application reads from the server, CT-Library discards all results generated by the canceled command.	Causes this connection's cursors to enter an undefined state. To determine the state of a cursor, an application can call vbct_cmd_props_num, and vbct_cmd_props_str with property as CS_CUR_STATUS.
CS_CANCEL_CURREN T	vbct_cancel discards the current result set.	Safe to use on connections with open cursors.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.

CS_FAIL	The routine failed.
CS_PENDING	Asynchronous network I/O is in effect. Refer to Asychronous Programming.
CS_CANCELED	The cancel operation was canceled. Only a CS_CANCEL_CURRENT type of cancel can be canceled.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.
CS_TRYING	A cancel operation is already pending for this connection.

Remarks

Canceling a command is equivalent to sending an attention to the server, instructing it to halt execution of the current command. When a command is canceled, any results generated by it are no longer available to an application.

Canceling results is equivalent to discarding a buffer's worth of results. Once results are canceled, they are no longer available to an application. If the result set has not been completely processed, subsequent results remain available.

Canceling a Command

To cancel the current command and all results generated by it, an application calls vbct_cancel with type as CS_CANCEL_ATTN or CS_CANCEL_ALL. Both of these calls tell CT-Library to do the following:

- Send an attention to the server, instructing it to halt execution of the current command.
- Discard any results already generated by the command.

Both types of cancels return CS_SUCCEED immediately, without sending an attention to the server, if no command is in progress.

If an application has not yet called vbct_send to send an initiated command or command batch:

- A CS_CANCEL_ALL cancel discards the initiated command or command batch without sending an attention to the server. A CS_CANCEL_ATTN cancel has no effect.

A connection can become unusable due to error. If this occurs, CT-Library marks the connection as "dead." An application can use the CS_CON_STATUS property to determine if a connection has been marked dead.

If a connection has been marked dead because of a results-processing error, an application can try calling vbct_cancel(CS_CANCEL_ALL or CS_CANCEL_ATTN) to "revive" the connection. If this fails, the application must close the connection and drop its CS_CONNECTION structure.

The difference between CS_CANCEL_ALL and CS_CANCEL_ATTN is:

- CS_CANCEL_ALL causes CT-Library to immediately discard the canceled command's results (if any).

- CS_CANCEL_ATTN causes CT-Library to wait until the application attempts to read from the server before discarding the results.

This difference is important because CT-Library must read from the result stream in order to discard results, and it is not always safe to read from the result stream.

It is not safe to read from the result stream from within callbacks or interrupt handlers, or when an asynchronous routine is pending. It is safe to read from the result stream anytime an application is running in its main-line code, except when an asynchronous operation is pending.

Use CS_CANCEL_ATTN from within callbacks or interrupt handlers, or when an asynchronous operation is pending.

Use CS_CANCEL_ALL in main-line code, except when an asynchronous operation is pending.

CS_CANCEL_ALL leaves the command structure in a "clean" state, available for use in another operation. When a command is canceled with CS_CANCEL_ATTN, however, the command structure cannot be reused until a CT-Library routine returns CS_CANCELED.

The CT-Library routines that can return CS_CANCELED are:

- vbct_cancel(CS_CANCEL_CURRENT)
- vbct_fetch
- vbct_get_data
- vbct_options_num
- vbct_options_str
- vbct_recvpassthru
- vbct_results
- vbct_send
- vbct_sendpassthru

CS_CANCEL_ATTN has two primary uses:

1. To cancel commands from within an application's interrupt handlers or callback routines.

2. In asynchronous applications, to cancel pending calls to the result-processing routines vbct_results and vbct_fetch.

Canceling Current Results

To cancel current results, an application calls vbct_cancel with type as CS_CANCEL_CURRENT. This tells CT-Library to discard the current results; it is equivalent to calling vbct_fetch until it returns CS_END_DATA.

The next buffer's worth of results, if any, remains available to the application, and the current command is not affected.

Canceling results clears the bindings between the result items and program variables.

A CS_CANCEL_CURRENT type of cancel is legal for all types of result sets, even those that contain no fetchable results. If a result set contains no fetchable results, a cancel has no effect.

See Also

vbct_fetch, vbct_results

vbct_capability

Set or retrieve a client/server capability.

Syntax

vbct_capability(connection&, action&, type&, capability&, value&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

action&

One of the following symbolic values:

Value of action:	vbct_capability:
CS_SET	Sets a capability.
CS_GET	Retrieves a capability.

type&

The type category of the capability. The following table lists the symbolic values that are legal for type:

Value of type:	What it means:
CS_CAP_REQUEST	Request capabilities. These capabilities describe the types of requests that a connection can support. Request capabilities are retrieve-only.
CS_CAP_RESPONSE	Response capabilities. These capabilities describe the types of responses that a server can send to a connection. An application can set response capabilities before a connection is open and can retrieve response capabilities at any time.

capability&

The capability of interest. The following two tables list the symbolic values that are legal for capability:

NOTE: In addition to the values listed in the tables, capability can have the special value CS_ALL_CAPS, to indicate that an application is setting or retrieving all response or request capabilities simultaneously. CS_ALL_CAPS is primarily of use in gateway applications. A typical CT-Library application will only need to set or retrieve a small number of capabilities.

CS_CAP_REQUEST Capabilities Table

CS_CAP_REQUEST Capability:	Meaning:	Capability relates to:
CS_CON_INBAND	In-band (non-expedited) attentions.	Connections
CS_CON_OOB	Out-of-band (expedited) attentions.	Connections
CS_CSR_ABS	Fetch of specified absolute cursor row.	Cursors
CS_CSR_FIRST	Fetch of first cursor row.	Cursors
CS_CSR_LAST	Fetch of last cursor row.	Cursors
CS_CSR_MULTI	Multi-row cursor fetch.	Cursors
CS_CSR_PREV	Fetch previous cursor row.	Cursors

CS_CSR_REL	Fetch specified relative cursor row.	Cursors
CS_DATA_BIN	Binary datatype.	Datatypes
CS_DATA_VBIN	Variable-length binary type.	Datatypes
CS_DATA_LBIN	Long binary datatype.	Datatypes
CS_DATA_BIT	Bit datatype.	Datatypes
CS_DATA_BITN	Nullable bit values.	Datatypes
CS_DATA_BOUNDARY	Secure Server boundary datatypes.	Datatypes
CS_DATA_CHAR	Character datatype.	Datatypes
CS_DATA_VCHAR	Variable-length character datatype.	Datatypes
CS_DATA_LCHAR	Long character datatype.	Datatypes
CS_DATA_DATE4	Short datetime datatype.	Datatypes
CS_DATA_DATE8	Datetime datatype.	Datatypes
CS_DATA_DATETIMEN	NULL datetime values.	Datatypes
CS_DATA_DEC	Decimal datatype.	Datatypes
CS_DATA_FLT4	4-byte float datatype.	Datatypes
CS_DATA_FLT8	8-byte float datatype.	Datatypes
CS_DATA_FLTN	Nullable float values.	Datatypes
CS_DATA_IMAGE	Image datatype.	Datatypes
CS_DATA_INT1	Tiny integer datatype.	Datatypes
CS_DATA_INT2	Small integer datatype.	Datatypes
CS_DATA_INT4	Integer datatype.	Datatypes
CS_DATA_INTN	NULL integers.	Datatypes
CS_DATA_MNY4	Short money datatype.	Datatypes
CS_DATA_MNY8	Money datatype.	Datatypes
CS_DATA_MONEYN	NULL money values.	Datatypes
CS_DATA_NUM	Numeric datatype.	Datatypes
CS_DATA_SENSITIVITY	Secure Server sensitivity datatypes.	Datatypes
CS_DATA_TEXT	Text datatype.	Datatypes
CS_OPTION_GET	Whether the client can get current option values from the server.	Options
CS_PROTO_BULK	Tokenized bulk copy.	Bulk copy
CS_PROTO_DYNAMIC	Descriptions for prepared statements come back at prepare time.	Dynamic SQL
CS_PROTO_DYNPROC	CT-Library prepends SQL to a Dynamic SQL prepare statement.	Dynamic SQL
CS_REQ_BCP	Bulk copy requests.	Commands
CS_REQ_CURSOR	Cursor requests.	Commands
CS_REQ_DYN	Dynamic SQL requests.	Commands
CS_REQ_LANG	Language requests.	Commands
CS_REQ_MSG	Message commands.	Commands
CS_REQ_MSTMT	Multiple server commands per CT-Library language command.	Commands
CS_REQ_NOTIF	Registered procedure notifications.	Commands
CS_REQ_PARAM	Use PARAM/PARAMFMT TDS streams for requests.	Commands
CS_REQ_URGNOTIF	Send notifications with the "urgent" bit set in the TDS packet header.	Registered procedures

CS_REQ_RPC	Remote procedure requests.	Commands

CS_CAP_RESPONSE Capabilities Table

CS_CAP_RESPONSE Capability:	**Meaning:**	**Capability relates to:**
CS_CON_NOINBAND	No in-band (non-expedited) attentions.	Connections
CS_CON_NOOOB	No out-of-band (expedited) attentions.	Connections
CS_DATA_NOBIN	No binary datatype.	Datatypes
CS_DATA_NOVBIN	No variable-length binary type.	Datatypes
CS_DATA_NOLBIN	No long binary datatype.	Datatypes
CS_DATA_NOBIT	No bit datatype.	Datatypes
CS_DATA_NOBOUNDARY	No Secure Server boundary datatypes.	Datatypes
CS_DATA_NOCHAR	No character datatype.	Datatypes
CS_DATA_NOVCHAR	No variable-length character datatype.	Datatypes
CS_DATA_NOLCHAR	No long character datatype.	Datatypes
CS_DATA_NODATE4	No short datetime datatype.	Datatypes
CS_DATA_NODATE8	No datetime datatype.	Datatypes
CS_DATA_NODATETIMEN	No NULL datetime values.	Datatypes
CS_DATA_NODEC	No decimal datatype.	Datatypes
CS_DATA_NOFLT4	No 4-byte float datatype.	Datatypes
CS_DATA_NOFLT8	No 8-byte float datatype.	Datatypes
CS_DATA_NOIMAGE	No image datatype.	Datatypes
CS_DATA_NOINT1	No tiny integer datatype.	Datatypes
CS_DATA_NOINT2	No small integer datatype.	Datatypes
CS_DATA_NOINT4	No integer datatype.	Datatypes
CS_DATA_NOINT8	No 8-byte integer datatype.	Datatypes
CS_DATA_NOINTN	No NULL integers.	Datatypes
CS_DATA_NOMNY4	No short money datatype.	Datatypes
CS_DATA_NOMNY8	No money datatype.	Datatypes
CS_DATA_NOMONEYN	No NULL money values.	Datatypes
CS_DATA_NONUM	No numeric datatype.	Datatypes
CS_DATA_NOSENSITIVITY	No Secure Server sensitivity datatypes.	Datatypes
CS_DATA_NOTEXT	No text datatype.	Datatypes
CS_RES_NOEED	No extended error results.	Results
CS_RES_NOMSG	No message results.	Results
CS_RES_NOPARAM	Don't use PARAM/PARAMFMT TDS streams for RPC results.	Results
CS_RES_NOSTRIPBLANKS	The server shouldn't strip blanks when returning data from nullable fixed- length character columns.	Results
CS_RES_NOTDSDEBUG	No TDS debug token in response to certain dbcc commands.	Results

value&

If a capability is being set, value points to a CS_BOOL variable that has the value CS_TRUE or

CS_FALSE.

If a capability is being retrieved, value points to a CS_BOOL-sized variable which vbct_capability sets to CS_TRUE or CS_FALSE.

CS_TRUE indicates that a capability is enabled. For example, if the CS_RES_NOEED capability is set to CS_TRUE, no extended error data will be returned on the connection.

NOTE: If capability is CS_ALL_CAPS, value must point to a CS_CAP_TYPE structure.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Capabilities describe client/server features that a connection supports.

There are two types of capabilities: CS_CAP_RESPONSE capabilities, also called "response capabilities," and CS_CAP_REQUEST capabilities, also called "request capabilities."

- An application uses request capabilities to determine what kinds of requests a server connection supports. For example, an application can retrieve the CS_REQ_CURSOR capability to find out whether a connection supports cursor requests.

- An application uses response capabilities to prevent the server from sending a type of response that the application cannot process. For example, an application can prevent a server from sending NULL money values by setting the CS_DATA_NOMONEYN response capability to CS_TRUE.

An application can do the following before a connection is open:

- Retrieve request or response capabilities, to determine what request and response features are normally supported at the application's current TDS (Tabular Data Stream) version level. An application's TDS level defaults to a value based on the CS_VERSION level that the application requested in its call to vbct_init.
- Set response capabilities, to indicate that a connection does not wish to receive particular types of server responses. Note that an application cannot set request capabilities, which are retrieve-only.

An application can do the following after a connection is open:

- Retrieve request capabilities to find out what types of requests the connection will support.
- Retrieve response capabilities to find out whether the server has agreed to withhold the previously-indicated response types from the connection.

Capabilities are determined by a connection's TDS version level. Not all TDS versions support the same capabilities. For example, 4.0 TDS does not support registered procedure notifications or cursor requests. 4.0 TDS does, however, support bulk copy requests, remote procedure call requests, row results, and compute row results. A connection's TDS version level is negotiated during the connection process.

If an application sets the CS_TDS_VERSION property, CT-Library overwrites existing capability values with default capability values corresponding to the new TDS version. For this reason, an application should set CS_TDS_VERSION before setting any capabilities for a connection.

Because CS_TDS_VERSION is a negotiated login property, the server can change its value at connection time. If this occurs, CT-Library will overwrite existing capability values with default capability values corresponding to the new TDS version.

Because capability values can change at connection time, an application must call vbct_capability after a connection is open in order to determine what capability values are in effect for the connection.

When a connection is closed, CT-Library resets its capability values to values corresponding to the application's default TDS version.

Setting and Retrieving Multiple Capabilities

Gateway applications often need to set or retrieve all capabilities of a type category with a single call to vbct_capability. To do this, an application calls vbct_capability with:

- type as the type category of interest
- capability as CS_ALL_CAPS
- value as a CS_CAP_TYPE structure

CT-Library provides the following macros to enable an application to set, clear, and test bits in a CS_CAP_TYPE structure:

- CS_SET_CAPMASK(mask, capability)
- CS_CLR_CAPMASK(mask, capability)
- CS_TST_CAPMASK(mask, capability)

where mask is a pointer to a CS_CAP_TYPE structure and capability is the capability of interest.

See Also

vbct_con_props_num, vbct_con_props_str, vbct_connect , vbct_options_num, vbct_options_str.

vbct_close

Close a server connection.

Syntax

vbct_close(connection&, option&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

option&

The option, if any, to use for the close. The following table lists the symbolic values that are legal for option:

Value of option:	**What it means:**
CS_UNUSED(10.0+ servers only)	Default behavior. vbct_close sends a logout message to the server and reads the response to this message before closing the connection. If the connection has results pending, vbct_close returns CS_FAIL.
CS_FORCE_CLOSE	The connection is closed whether or not results are pending, and without notifying the server. This option is primarily for use when an application is hung waiting for a server response. It is also useful if vbct_results, vbct_fetch, or vbct_cancel returns CS_FAIL

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_PENDING	Asynchronous network I/O is in effect. Refer to Asychronous Programming. If asynchronous network I/O is in effect and vbct_close is called with option as CS_FORCE_CLOSE, it returns CS_SUCCEED or CS_FAIL immediately to indicate the network response. In this case, no completion callback event occurs.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming. Note that vbct_close does not return CS_BUSY when called with option as CS_FORCE_CLOSE.

The most common reason for a vbct_close(CS_UNUSED) failure is pending results on the connection.

Remarks

To de-allocate a CS_CONNECTION, an application can call vbct_con_drop after the connection has been successfully closed.

A connection can become unusable due to error. If this occurs, CT-Library marks the connection as "dead". An application can use the CS_CON_STATUS property to determine if a connection has been marked dead.

If a connection has been marked dead, an application must call vbct_close(CS_FORCE_CLOSE) to close the connection and vbct_con_drop to drop its CS_CONNECTION structure.

An exception to this rule occurs for certain types of results-processing errors. If a connection is marked dead while processing results, the application can try calling vbct_cancel(CS_CANCEL_ALL or CS_CANCEL_ATTN) to "revive" the connection. If this fails, the application must close the connection and drop its CS_CONNECTION structure.

If the connection is using asynchronous network I/O, vbct_close returns CS_PENDING. When the server response arrives, CT-Library closes the connection and then calls the completion callback installed for the connection, if any.

The behavior of vbct_close depends on the value of option, which determines the type of close. Each section below contains information on a type of close.

Default Close Behavior

When connected to a 10.0+ server, vbct_close sends a logout message to the server and reads the response to this message before terminating the connection. The contents of this message do not affect vbct_close's behavior.

An application cannot call vbct_close(CS_UNUSED) when an asynchronous operation is pending.

CS_FORCE_CLOSE Behavior

The connection is closed whether or not it has an open cursor or pending results.

vbct_close does not behave asynchronously when called with the CS_FORCE_CLOSE option. When vbct_close(CS_FORCE_CLOSE) is called to close an asynchronous connection, it returns CS_SUCCEED or CS_FAIL immediately, to indicate the network response. In this case, no completion callback event occurs.

CS_FORCE_CLOSE is useful when:

- A connection has been marked as dead.
- An application is hung, waiting for a server response.
- An application cannot call vbct_close(CS_UNUSED) because results are pending.

Because no logout message is sent to the server, the server cannot tell whether the close is intentional or whether it is the result of a lost connection or crashed client.

An application can call vbct_close(CS_FORCE_CLOSE) when an asynchronous operation is pending.

See Also

vbct_install_callbacks, vbct_con_drop, vbct_connect, vbct_con_props_num, vbct_con_props_str

vbct_cmd_alloc

Allocate a CS_COMMAND structure.

Syntax

vbct_cmd_alloc(connection&, command&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

command&

The parameter is primed with the command pointer.   This command pointer is a parameter used in functions such as vbct_send.

In case of error, vbct_cmd_alloc sets command to NULL.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

The most common reason for a vbct_cmd_alloc failure is a lack of adequate memory.

Remarks

A CS_COMMAND structure, also called a "command structure," is a control structure that a CT-Library application uses to send commands to a server and process the results of those commands.

An application must call vbct_con_alloc to allocate a connection structure before calling vbct_cmd_alloc to allocate command structures for the connection.

However, it is not necessary that the connection structure represent an open connection.   (An application opens a connection by calling vbct_connect to connect to a server.)

See Also

vbct_command_num, vbct_command_str, vbct_cmd_drop, vbct_cmd_props_num, vbct_cmd_props_str, vbct_con_alloc

vbct_cmd_drop

De-allocate a CS_COMMAND structure.

Syntax

vbct_cmd_drop(command&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

vbct_cmd_drop returns CS_FAIL if one of the following occurs:

- command has an active command. A command that has been initialized but not yet sent is considered to be active.
- command has pending results.

Remarks

A CS_COMMAND structure is a control structure that a CT-Library application uses to send commands to a server and process the results of those commands.

Once a command structure has been de-allocated, it cannot be re-used. To allocate a new CS_COMMAND structure, an application can call vbct_cmd_alloc.

Before de-allocating a command structure, an application should cancel any active commands, process or cancel any pending results.

See Also

vbct_command_num, vbct_command_str, vbct_cmd_alloc

vbct_cmd_props_num

Set or retrieve command structure properties.

Syntax

vbct_cmd_props_num(command&, action&, property&, buffer&, outlen&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

action&

One of the following symbolic values:

Value of action:	**vbct_cmd_props_num:**
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its CT-Library default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer&

If a property value is being set, buffer points to the value to use in setting the property.

If a property value is being retrieved, buffer points to the variable in which vbct_cmd_props_num will place the requested information.

outlen&

A long variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbct_cmd_props_num sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming

Remarks

Command structure properties affect the behavior of an application at the command structure level.

All command structures allocated for a connection pick up default property values from the parent connection. An application can override these default values by calling vbct_cmd_props_num and vbct_cmd_props_str.

If an application changes connection property values after allocating command structures for the connection, the existing command structures will not pick up the new property values. New command structures allocated for the connection will use the new property values as defaults.

An application can use vbct_cmd_props_num to set or retrieve the following properties:

Property name:	**What it is:**	**buffer is:**	**CT-Library can set or retrieve at what level?**	**Notes**
CS_CUR_ID	The cursor's identification number.	Set to an integer value.	Command.	Retrieve only, after CS_CUR_STATUS indicates an existing cursor.
CS_CUR_ROWCOUNT	The current value of cursor rows. Cursor rows is the number of rows returned to CT-Library per internal fetch request.	Set to an integer value.	Command.	Retrieve only, after CS_CUR_STATUS indicates an existing cursor.
CS_CUR_STATUS	The cursor's status.	Set to a CS_INT- sized bit-mask.	Command.	Retrieve only.
CS_HIDDEN_KEYS	Whether or not to expose hidden keys.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	Context, connection, command.	Cannot be set at the command level if results are pending or a cursor is open.
CS_PARENT_HANDLE	The address of the command structure's parent connection.	Set to an address.	Connection, command.	Retrieve only.
CS_USERDATA	User-allocated data.	User-allocated data.	Connection, command. To set CS_USERDATA at the context level, call vbcs_config.	None

See Also

vbct_config_num, vbct_config._str, vbct_cmd_alloc, vbct_con_props_num, vbct_con_props_str, vbct_res_info

vbct_cmd_props_str

Set or retrieve command structure properties.

Syntax

vbct_cmd_props_str(command&, action&, property&, buffer$, outlen&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

action&

One of the following symbolic values:

Value of action:	**vbct_cmd_props_str:**
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its CT-Library default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer$

If a property value is being set, buffer points to the value to use in setting the property.

If a property value is being retrieved, buffer points to the variable in which vbct_cmd_props_str will place the requested information.

outlen&

A long variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbct_cmd_props_str sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Command structure properties affect the behavior of an application at the command structure level.

All command structures allocated for a connection pick up default property values from the parent connection. An application can override these default values by calling vbct_cmd_props_num and vbct_cmd_props_str.

If an application changes connection property values after allocating command structures for the connection, the existing command structures will not pick up the new property values. New command structures allocated for the connection will use the new property values as defaults.

An application can use vbct_cmd_props_str to set or retrieve the following properties:

Property name:	**What it is:**	**buffer is:**	**CT-Library can set or retrieve at what level?**	**Notes**
CS_CUR_NAME	The cursor's name, as defined in an application's ct_cursor(CS_CURSOR_DECLARE) call.	Set to a null-terminated character string.	Command.	Retrieve only, after ct_cursor CS_CUR_STATUS returns CS_SUCCEED.
CS_USERDATA	User-allocated data.	User-allocated data.	Connection, command. To set CS_USERDATA at the context level, call vbcs_config.	None

See Also

vbct_config_num, vbct_config_str, vbct_cmd_alloc, vbct_con_props_num, vbct_con_props_str, vbct_res_info

vbct_command_num

Function

Initiate a language, package, RPC, message, or send-data command.

Syntax

vbct_command_num(command&, type&, buffer&, option&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

type&

The type of command to initiate. The table in the Summary of Parameters table below lists the symbolic values that are legal for type.

buffer&

A pointer to data space.

option&

The option associated with this command, if any.

Currently, RPC (remote procedure call), send-data, and send-bulk-data commands take options. For all other types of commands, pass option as CS_UNUSED.

The following table lists the symbolic values that are legal for option:

type is:	**Value of option:**	**Meaning:**
CS_RPC_CMD	CS_RECOMPILE	Recompile the stored procedure before executing it.
	CS_NORECOMPILE	Do not recompile the stored procedure before executing it.
	CS_UNUSED	Equivalent to S_NORECOMPILE.
CS_SEND_DATA_CMD	CS_COLUMN_DATA	The data will be used for a text or image column update.
	CS_BULK_DATA	For internal Sybase use only. The data will be used for a bulk copy operation.
CS_SEND_BULK_ CMD	CS_BULK_INIT	For internal Sybase use only. Initialize a bulk copy operation.
	CS_BULK_CONT	For internal Sybase use only. Continue a bulk copy operation.

Summary of Parameters Table

Value of type:	**vbct_command_num initiates:**	**buffer is:**	**buflen is:**
CS_MSG_CMD	A message command.	A pointer to a CS_SMALLINT	CS_UNUSED

		representing the message id.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Initiating a command is the first step in sending it to a server.

Follow these steps to send a command to a server:

- Initiate the command by calling vbct_command_num. This routine sets up internal structures that are used in building a command stream to send to the server.
- Pass parameters for the command (if required) by calling vbct_param once for each parameter that the command requires.

Not all commands require parameters. For example, a remote procedure call command may or may not require parameters, depending on the stored procedure being called.

- Send the command to the server by calling vbct_send.
- Verify the success of the command by calling vbct_results.

This last step does not imply that an application need only call vbct_results once. An application needs to continue calling vbct_results until it no longer returns CS_SUCCEED.

An application can call vbct_cancel with type as CS_CANCEL_ALL to clear a command that has been initiated but not yet sent.

Within a single connection, the following rules apply to the use of vbct_command_num:

- After calling vbct_command_num to initiate a command, an application must either send the initiated command or clear it before calling vbct_command_num a second time.
- After sending a command initiated via vbct_command_num, an application must completely process or cancel all results generated by the command before calling vbct_command_num to initiate another command.

Each section below contains information on one of the types of commands that vbct_command_num can initiate.

Message Commands

Message commands and results provide a way for clients and servers to communicate specialized information to one another.

A message has an "id", which an application provides via vbct_command_num's buffer parameter.

Ids for user-defined messages must be greater than or equal to CS_USER_MSGID and less than or equal to CS_USER_MAX_MSGID.

If a message requires parameters, the application can call vbct_param once for each parameter that the message requires.

Send-Data Commands

An application uses a send-data command to write large amounts of text or image data to a server.

An application typically calls:

- vbct_command_num to initiate the send-data command.
- vbct_data_info to set the I/O descriptor for the operation.
- vbct_send_data to write the value, in chunks, to the data stream.
- vbct_send to send the command to the server.

See Also

vbct_cmd_alloc, vbct_dynamic, vbct_param, vbct_send

vbct_command_str

Initiate a language, package, RPC, message, or send-data command.

Syntax

vbct_command_str(command&, type&, buffer$, option&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

type&

The type of command to initiate. The table in the Summary of Parameters section below lists the symbolic values that are legal for type.

buffer$

A pointer to data space.

option&

The option associated with this command, if any.

Currently, RPC (remote procedure call), send-data, and send-bulk-data commands take options. For all other types of commands, pass option as CS_UNUSED.

The following table lists the symbolic values that are legal for option:

type is:	Value of option:	Meaning:
CS_RPC_CMD	CS_RECOMPILE	Recompile the stored procedure before executing it.
	CS_NORECOMPILE	Do not recompile the stored procedure before executing it.
	CS_UNUSED	Equivalent to CS_NORECOMPILE.
CS_SEND_DATA_CMD	CS_COLUMN_DATA	The data will be used for a text or image column update.
	CS_BULK_DATA	For internal Sybase use only. The data will be used for a bulk copy operation.
CS_SEND_BULK_CMD	CS_BULK_INIT	For internal Sybase use only. Initialize a bulk copy operation.
	CS_BULK_CONT	For internal Sybase use only. Continue a bulk copy operation.

Summary of Parameters Table

Value of type:	vbct_command_str initiates:	buffer is:	buflen is:
CS_LANG_CMD	A language command.	A pointer to the text of the language command.	The length of the buffer data or CS_NULLTERM.
CS_PACKAGE_CMD	A package command.	A pointer to the name of the package.	The length of the buffer data or

			CS_NULLTERM.
CS_RPC_CMD	A remote procedure call command.	A pointer to the name of the remote procedure.	The length of the buffer data or CS_NULLTERM.
CS_SEND_BULK_CMD	A SYBASE internal send-bulk-data command.	A pointer to the database table name.	The length of the buffer data or CS_NULLTERM.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Initiating a command is the first step in sending it to a server.

Follow these steps to send a command to a server:

- Initiate the command by calling vbct_command_str. This routine sets up internal structures that are used in building a command stream to send to the server.
- Pass parameters for the command (if required) by calling vbct_param once for each parameter that the command requires.

Not all commands require parameters. For example, a remote procedure call command may or may not require parameters, depending on the stored procedure being called.

- Send the command to the server by calling vbct_send.
- Verify the success of the command by calling vbct_results.

This last step does not imply that an application need only call vbct_results once. An application needs to continue calling vbct_results until it no longer returns CS_SUCCEED.

An application can call vbct_cancel with type as CS_CANCEL_ALL to clear a command that has been initiated but not yet sent.

Within a single connection, the following rules apply to the use of vbct_command_str:

- After calling vbct_command_str to initiate a command, an application must either send the initiated command or clear it before calling vbct_command_str a second time.
- After sending a command initiated via vbct_command_str, an application must completely process or cancel all results generated by the command before calling vbct_command_str to initiate another command.

Each section below contains information on one of the types of commands that vbct_command_str can initiate.

Language Commands

Language commands contain a character string that represents one or more commands in a server's own language.

A language command can be in any language, so long as the server to which it is directed can understand it. SQL Server understands Transact-SQL, but an Open Server application constructed with SYBASE Server-Library can be written to understand any language.

If the language command string contains host variables, an application can pass values for these variable by calling vbct_param once for each variable that the language string contains.

Transact-SQL command variables must begin with a colon (:).

Package Commands

A package command instructs an IBM DB/2 database server to execute a package.   A package is similar to a remote procedure.   It contains precompiled SQL statements that are executed as a unit when the package is invoked.

If the package requires parameters, the application can call vbct_param once for each parameter that the package requires.

RPC (remote procedure call) Commands

An RPC (remote procedure call) command instructs a server to execute a stored procedure either on this server or a remote server.

An application initiates an RPC command by calling vbct_command_str with buffer as the name of the stored procedure to execute.

If an application is using an RPC command to execute a stored procedure that requires parameters, the application can call vbct_param once for each parameter the stored procedure requires.

After sending an RPC command with vbct_send , an application can process the stored procedure's results with vbct_results and vbct_fetch. vbct_results and vbct_fetch are used to process both the result rows generated by the stored procedure and the return parameters and status from the procedure, if any.

An alternative way to call a stored procedure is by executing a language command containing a Transact-SQL execute statement.

Send-Bulk-Data Commands

Internally, SYBASE uses send-bulk-data commands as part of its implementation of CT-Library's bulk copy routines

See Also

vbct_cmd_alloc, vbct_dynamic, vbct_param, vbct_send

vbct_con_alloc

Allocate a CS_CONNECTION structure.

Syntax

vbct_con_alloc(context&, connection&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbct_ctx_alloc function.

connection&

The parameter is primed with the connection pointer. This connection pointer is a parameter used in functions such as vbct_con_connect.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a vbct_con_alloc failure is a lack of adequate memory.

Remarks

A CS_CONNECTION structure, also called a "connection structure," contains information about a particular client/server connection.

Before calling vbct_con_alloc, an application must allocate a context structure by calling the CT-Library routine vbcs_ctx_alloc, and must initialize CT-Library by calling vbct_init.

Follow these steps to connect to a server:

- Calls vbct_con_alloc to allocate a CS_CONNECTION structure.
- Calls vbct_con_props_num and vbct_con_props_str to set the values of connection-specific properties.
- Calls vbct_connect to create the connection and log in to the server.

An application can have multiple open connections to one or more servers at the same time.

For example, an application can simultaneously have two connections to the server ALPHA, one connection to BETA, and one connection to SIGMA. The context property CS_MAX_CONNECT, set by vbct_config, determines the maximum number of open connections allowed per context.

Each server connection requires a separate CS_CONNECTION structure.

In order to send commands to a server, one or more command structures must be allocated for a connection. vbct_cmd_alloc allocates a command structure.

See Also

vbcs_ctx_alloc, vbct_cmd_alloc, vbct_close, vbct_connect, vbct_con_props_num, vbct_con_props_str.

vbct_con_drop

De-allocate a CS_CONNECTION structure.

Syntax

vbct_con_drop(connection&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

The most common reason for a vbct_con_drop failure is that the connection is still open.

Remarks

When a CS_CONNECTION structure is de-allocated, all CS_COMMAND structures associated with it are de-allocated.

A CS_CONNECTION structure contains information about a particular client/server connection.

Once a CS_CONNECTION has been de-allocated, it cannot be reused. To allocate a new CS_CONNECTION, an application can call vbct_con_alloc.

An application cannot de-allocate a CS_CONNECTION structure until the connection it represents is closed. To close a connection, an application can call vbct_close.

A connection can become unusable due to error. If this occurs, CT-Library marks the connection as "dead". An application can use the CS_CON_STATUS property to determine if a connection has been marked dead.

If a connection has been marked dead, an application must call vbct_close(CS_FORCE_CLOSE) to close the connection and vbct_con_drop to drop its CS_CONNECTION structure.

An exception to this rule occurs for certain types of results-processing errors. If a connection is marked dead while processing results, the application can try calling vbct_cancel(CS_CANCEL_ALL or CS_CANCEL_ATTN) to "revive" the connection. If this fails, the application must close the connection and drop its CS_CONNECTION structure.

See Also

vbct_con_alloc, vbct_close, vbct_connect, vbct_con_props_num, vbct_con_props_str.

vbct_con_props_num

Set or retrieve connection structure properties.

Syntax

vbct_con_props_num(connection&, action&, property&, buffer&, outlen&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

action&

One of the following symbolic values:

Value of action:	**vbct_con_props_num**
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its CT-Library default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer&

If a property value is being set, buffer contains the value to use in setting the property.

If buffer is a fixed-length or symbolic value, pass buflen as CS_UNUSED.

outlen&

A pointer to an integer variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbct_con_props_num sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Connection properties define aspects of CT-Library behavior at the connection level.

All connections created within a context pick up default property values from the parent context. An application can override these default values by calling vbct_con_props_num and vbct_con_props_str.

If an application changes context property values after allocating connections for the context, existing connections will not pick up the new property values. New connections allocated within the context will use the new property values as defaults.

All command structures allocated for a connection pick up default property values from the parent connection. An application can override these default values by calling vbct_cmd_props_num and vbct_cmd_props_str to set property values at the command structure level.

If an application changes connection property values after allocating command structures for the connection, the existing command structures will not pick up the new property values. New command structures allocated for the connection will use the new property values as defaults.

Some connection properties only take effect if they are set before an application calls vbct_connect to establish the connection. See the "Notes" column in the CT-Library connection properties Table below.

An application can use vbct_con_props_num to set or retrieve the following properties:

Property name:	**What it is:**	**buffer is:**	**CT-Library can set or retrieve at what level?**	**Notes**
CS_ANSI_BINDS	Whether or not to use ANSI-style binds.	CS_TRUE or CS_FALSE.	Context, connection.	
CS_ASYNC_NOTIFS	Whether a connection will receive registered procedure notifications asynchronously.	CS_TRUE or CS_FALSE.	Connection.	Login property. Cannot be set after connection is established.
CS_BULK_LOGIN	Whether or not a connection is enabled to perform bulk copy "in" operations.	CS_TRUE or CS_FALSE.	Connection.	
CS_CHARSETCNV	Whether or not character set conversion is taking place.	CS_TRUE or CS_FALSE.	Connection.	Retrieve only, after connection is established.
CS_COMMBLOCK	A pointer to a communication sessions block. This property is specific to IBM-370 systems and is ignored by all other platforms.	A pointer value.	Connection.	Cannot be set after connection is established.
CS_CON_STATUS	The connection's status.	A CS_INT-sized bit-mask.	Connection.	Retrieve only.
CS_DIAG_TIMEOUT	When in-line error handling is in effect, whether CT-Library should fail or retry on timeout errors.	CS_TRUE or CS_FALSE.	Connection.	
CS_DISABLE_POLL	Whether or not to disable polling. If	CS_TRUE or CS_FALSE.	Context,	Useful in layered

	polling is disabled, vbct_poll does not report asynchronous operation completions.		connection.	asynchronous applications.
CS_EED_CMD	A pointer to a command structure containing extended error data.	A pointer value.	Connection.	Retrieve only.
CS_ENDPOINT	The file descriptor for a connection.	An integer value, or -1 if the platform does not support CS_END POINT.	Connection.	Retrieve only, after connection is established.
CS_EXPOSE_FMTS	Whether or not to expose results of type CS_ROWFMT_RESULT and CS_COMPUTEFMT_RE SULT.	CS_TRUE or CS_FALSE.	Context, connection.	Cannot be set after connection is established.
CS_EXTRA_INF	Whether or not to return the extra information that's required when processing CT-Library messages in-line using a SQLCA, SQLCODE, SQLSTATE.	CS_TRUE or CS_FALSE.	Context, connection.	
CS_HIDDEN_KEYS	Whether or not to expose hidden keys.	CS_TRUE or CS_FALSE.	Context, connection, command.	
CS_LOC_PROP	A CS_LOCALE structure that defines localization information.	A CS_LOCALE structure previously allocated by the application.	Connection.	Login property. Cannot be set after connection is established.
CS_LOGIN_STATUS	Whether or not the connection is open.	CS_TRUE or CS_FALSE.	Connection.	Retrieve only.
CS_NETIO	Whether network I/O is synchronous or asynchronous.	CS_SYNC_IO, CS_ASYNC_IO.	Context, connection.	Asynchronous connections are either fully or deferred asynchronous to match their parent context.
CS_NOTIF_CMD	A pointer to a command structure containing registered procedure notification parameters.	A pointer value.	Connection.	Retrieve only.
CS_PACKETSIZE	The TDS packet size.	An integer value.	Connection.	Negotiated login property.

				Cannot be set after connection is established.
CS_PARENT_HANDLE	The address of the connection structure's parent context.	Set to an address.	Connection, command.	Retrieve only.
CS_SEC_APPDEFINED	Whether or not the connection will use application-defined challenge/response security handshaking.	CS_TRUE or CS_FALSE.	Connection.	Cannot be set after connection is established.
CS_SEC_CHALLENGE	Whether or not the connection will use Sybase-defined challenge/response security handshaking.	CS_TRUE or CS_FALSE.	Connection.	Cannot be set after connection is established.
CS_SEC_ENCRYPTION	Whether or not the connection will use encrypted password security handshaking.	CS_TRUE or CS_FALSE.	Connection.	Cannot be set after connection is established.
CS_SEC_ NEGOTIATE	Whether or not the connection will use trusted-user security handshaking.	CS_TRUE or CS_FALSE.	Connection.	Cannot be set after connection is established.
CS_TDS_VERSION	The version of the TDS protocol that the connection is using.	A symbolic version level.	Connection.	Negotiated login property. Cannot be set after connection is established.
CS_TEXTLIMIT	The largest text or image value to be returned on this connection.	An integer alue.	Context, connection.	
CS_USERDATA	User-allocated data.	User-allocated data.	Connection, command.	
CS_USERNAME	The name used to log into the server.	A character string.	Connection.	Login property. Cannot be set after connection is established.

See Also

vbct_capability, vbct_cmd_props_num, vbct_cmd_props_str, vbct_connect, vbct_config_num, vbct_config_str, vbct_init.

vbct_con_props_str

Set or retrieve connection structure properties.

Syntax

vbct_con_props_str(connection&, action&, property&, buffer$, outlen&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

action&

One of the following symbolic values:

Value of action:	**vbct_con_props_str**
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its CT-Library default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer$

If a property value is being set, buffer points to the value to use in setting the property.

If a property value is being set and the value in buffer is null-terminated, pass buflen as CS_NULLTERM.

If buffer is a fixed-length or symbolic value, pass buflen as CS_UNUSED.

outlen&

A pointer to an integer variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbct_con_props_str sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Connection properties define aspects of CT-Library behavior at the connection level.

All connections created within a context pick up default property values from the parent context. An application can override these default values by calling vbct_con_props_num and vbct_con_props_str.

If an application changes context property values after allocating connections for the context, existing connections will not pick up the new property values. New connections allocated within the context will use the new property values as defaults.

All command structures allocated for a connection pick up default property values from the parent connection. An application can override these default values by calling vbct_cmd_props_num and vbct_cmd_props_str to set property values at the command structure level.

If an application changes connection property values after allocating command structures for the connection, the existing command structures will not pick up the new property values. New command structures allocated for the connection will use the new property values as defaults.

Some connection properties only take effect if they are set before an application calls vbct_connect to establish the connection. See the "Notes" column in CT-Library connection properties Table below.

An application can use vbct_con_props_str to set or retrieve the following properties:

Property name:	**What it is:**	**buffer is:**	**CT-Library can set or retrieve at what level?**	**Notes**
CS_APPNAME	The application name used when logging into the server.	A character string	Connection	Login property. Cannot be set after connection is established.
CS_HOSTNAME	The host machine name.	A character string.	Connection.	Login property. Cannot be set after connection is established.
CS_PASSWORD	The password used to log into the server.	A character string.	Connection.	Login property.
CS_SERVERNAME	The name of the server to which this connection is connected.	A string value.	Connection.	Retrieve only after connection is established.
CS_TRANSACTION_ NAME	A transaction name.	A string value.	Connection.	

See Also

vbct_capability, vbct_cmd_props_num, vbct_cmd_props_str, vbct_connect, vbct_config_num, vbct_config_str, vbct_init

vbct_compute_info

Retrieve compute result information.

Syntax

vbct_compute_info(command&, type&, column&, buffer(Any), buflen&, outlen&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

type&

The type of information to return.

column&

The number of the compute column of interest, as it appears in the compute row result set. Compute columns appear in the order in which they are listed in the compute clause of a select statement. The first column is number 1, the second number 2, and so on.

buffer(Any)

A pointer to the space in which vbct_compute_info will place the requested information.

If buflen indicates that buffer is not large enough to hold the requested information, vbct_compute_info returns CS_FAIL.

buflen&

The length, in bytes, of the buffer data space or CS_UNUSED if buffer represents a fixed-length or symbolic value.

outlen&

A pointer to an integer variable.

vbct_compute_info sets outlen to the length, in bytes, of the requested information.

Summary of Parameters Table

Value of type:	Value of colnum:	vbct_compute_ info returns:	buffer is set to:	outlen is set to:
CS_BYLIST_LEN	CS_UNUSED	The number of elements in the bylist array.	An integer value.	sizeof(CS_INT)
CS_COMP_BYLIST	CS_UNUSED	An array containing the bylist that produced this compute row.	An array of CS_SMALLINT values.	The length of the array, in bytes.
CS_COMP_COLID	The column number of the compute column.	The select-list column id of the column from which the compute column derives.	An integer value.	sizeof(CS_INT)
CS_COMP_ID	CS_UNUSED	The compute id for	An integer value.	sizeof(CS_INT)

		the current compute row.		
CS_COMP_OP	The column number of the compute column.	The aggregate operator type for the compute column.	A symbolic value, one of: CS_OP_SUM CS_OP_AVG CS_OP_COUNT CS_OP_MIN CS_OP_MAX	sizeof(CS_INT)

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

Remarks

Compute rows result from the compute clause of a select statement. A compute clause generates a compute row every time the value of its by column-list changes. A compute row will contain one column for each aggregate operator in the compute clause. If a select statement contains multiple compute clauses, separate compute rows are generated by each clause.

Each compute row returned by the server is considered to be a distinct result set. This means each result set of type CS_COMPUTE_RESULT will contain exactly one row.

It is only legal to call vbct_compute_info when compute information is available; that is, after vbct_results returns CS_COMPUTE_RESULT or CS_COMPUTEFMT.

Each section below contains information about a particular type of compute result information.

The Bylist for a Compute Row

A select statement's compute clause may contain the keyword by, followed by a list of columns. This list, called the "bylist," divides the results into subgroups, based on changing values in the specified columns. The compute clause's aggregate operators are applied to each subgroup, generating a compute row for each subgroup.

The Select-List Column ID for a Compute Column

The select-list column id for a compute column is the select-list id of the column from which the compute column derives.

The Compute ID for this Compute Row

A SQL select statement can have multiple compute clauses, each of which returns a separate compute row. The compute id corresponding to the first compute clause in a select statement is 1.

The Aggregate Operator for a Particular Compute Row Column

When called with type as CS_COMP_OP, vbct_compute_info sets buffer to one of the following aggregate operator types:

buffer set to:	To indicate:
CS_OP_AVG	Average aggregate operator.

CS_OP_COUNT	Count aggregate operator.
CS_OP_MAX	Maximum aggregate operator.
CS_OP_MIN	Minimum aggregate operator.
CS_OP_SUM	Sum aggregate operator.

See Also

vbct_describe, vbct_res_info, vbct_results

vbct_config_num

Set or retrieve context properties.

Syntax

vbct_config_num(connection&, action&, property&, buffer(Any), outlen&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

action&

One of the following symbolic values:

Value of action:	**vbct_config_num:**
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its CT-Library default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer(Any)

If a property value is being set, buffer points to the value to use in setting the property.

If a property value is being retrieved, buffer points to the space in which vbct_config_num will place the requested information.

outlen&

A pointer to an integer variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbct_config_num sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Remarks

Context properties define aspects of CT-Library behavior at the context level.

All connections created within a context pick up default property values from the parent context. An application can override these default values by calling vbct_con_props_num and vbct_con_props_str to set property values at the connection level.

If an application changes context property values after allocating connections for the context, existing connections will not pick up the new property values. New connections allocated within the context will use the new property values as defaults.

There are three kinds of context properties:

1. Context properties specific to CT-Library.
2. Context properties specific to CT-Library.
3. Context properties specific to Server-Library.

vbcs_config_num sets and retrieves the values of CT-Library-specific context properties.

vbct_config_num sets and retrieves the values of CT-Library-specific context properties.

An application can use vbct_config_num to set or retrieve the following properties:

Property name:	**What it is:**	**buffer is:**	**CT-Library can set or retrieve at what level?**	**Notes**
CS_ANSI_BINDS	Whether or not to use ANSI-style binds.	CS_TRUE or CS_FALSE.	Context connection.	
CS_DISABLE_POLL	Whether or not to disable polling. If polling is disabled, vbct_poll does not report asynchronous operation completions.	CS_TRUE or CS_FALSE.	Context, connection	Useful in layered asynchronous applications.
CS_EXPOSE_FMTS	Whether or not to expose results of type CS_ROWFMT_RESULT and CS_COMPUTEFMT_RESULT.	CS_TRUE or CS_FALSE.	Context, connection	Takes effect only if set before connection is established.
CS_EXTRA_INF	Whether or not to return the extra information that's required when processing CT-Library messages in-line using a SQLCA, SQLCODE, or SQLSTATE.	CS_TRUE or CS_FALSE.	Context, connection	
CS_HIDDEN_KEYS	Whether or not to expose hidden keys.	CS_TRUE or CS_FALSE.	Context, connection command.	
CS_LOGIN_TIMEOUT	The login timeout value.	An integer value.	Context.	

CS_MAX_CONNECT	The maximum number of connections for this context.	An integer value.	Context.	
CS_NETIO	Whether network I/O is synchronous, fully asynchronous, or deferred asynchronous.	CS_SYNC_IO, CS_ASYNC_IO, or CS_DEFER_IO.	Context, connection.	Cannot be set for a context with open connections.
CS_NO_TRUNCATE	Whether CT-Library should truncate or sequence messages that are longer than CS_MAX_MSG.	CS_TRUE or CS_FALSE.	Context.	
CS_NOINTERRUPT	Whether or not the application can be interrupted.	CS_TRUE or CS_FALSE.	Context.	
CS_TEXTLIMIT	The largest text or image value to be returned on this connection.	An integer value.	Context, connection.	
CS_TIMEOUT	The timeout value.	An integer value.	Context.	
CS_VERSION	The version of CT-Library in use by this context.	A symbolic version level. Currently, the only possible value is CS_VERSION_100	Context.	Retrieve only.

See Also

vbct_cmd_props_num, vbct_cmd_props_str, vbct_capability, vbct_con_props_num, vbct_con_props_str, vbct_connect, vbct_init.

vbct_config_str

Set or retrieve context properties.

Syntax

vbct_config_str(connection&, action&, property&, buffer$, outlen&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

action&

One of the following symbolic values:

Value of action:	vbct_config_str:
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its CT-Library default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer$

If a property value is being set, buffer points to the value to use in setting the property.

If a property value is being retrieved, buffer points to the space in which vbct_config_str will place the requested information.

outlen&

A pointer to an integer variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbct_config_str sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Remarks

Context properties define aspects of CT-Library behavior at the context level.

All connections created within a context pick up default property values from the parent context. An application can override these default values by calling vbct_con_props_num and vbct_con_props_str to set property values at the connection level.

If an application changes context property values after allocating connections for the context, existing connections will not pick up the new property values.   New connections allocated within the context will use the new property values as defaults.

There are three kinds of context properties:

1. Context properties specific to CT-Library.
2. Context properties specific to CT-Library.
3. Context properties specific to Server-Library.

vbcs_config_str sets and retrieves the values of CT-Library-specific context properties.

vbct_config_str sets and retrieves the values of CT-Library-specific context properties.

An application can use vbct_config_str to set or retrieve the following properties:

Property name:	**What it is:**	**buffer is:**	**CT-Library can set or retrieve at what level?**	**Notes**
CS_IFILE	The path and name of the interfaces file.	A character string.	Context.	
CS_VER_STRING	CT-Library's true version string.	A character string.	Context.	Retrieve only.

See Also

vbct_cmd_props_num, vbct_cmd_props_str, vbct_capability, vbct_con_props_num, vbct_con_props_str, vbct_connect, vbct_init

vbct_connect

Connect to a server.

Syntax

vbct_connect(connection&, server_name$)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

Use vbct_con_alloc to allocate a CS_CONNECTION structure, and vbct_con_props_num and vbct_con_props_str to initialize this structure with login parameters.

server_name$

A pointer to the name of the server to connect to. server_name is the name given to the server in the interfaces file on the application's host machine. vbct_connect looks up server_name in the interfaces file to determine how to connect to this server.

If server_name is NULL, vbct_connect looks up the interfaces entry that corresponds to the value of the DSQUERY environment variable or logical name.   If DSQUERY has not been explicitly set, it has a value of "SYBASE".

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Common reason for a vbct_connect failure include:

- Unable to allocate sufficient memory.
- The maximum number of connections is already established. vbct_config_num and vbct_config_str are used to set the maximum number of connections allowed per context.
- Unable to open socket.
- Server name not found in interfaces file.
- Unknown host machine name.
- SQL Server is unavailable or does not exist.
- Login incorrect.
- Could not open interfaces file.

When vbct_connect returns CS_FAIL, it generates a CT-Library error number that indicates the error.

Remarks

Information about the connection is stored in a CS_CONNECTION structure, which uniquely identifies the connection. In the process of establishing a connection, vbct_connect sets up communication with the network, logs into the server, and communicates any connection-specific property information to the server.

Because creating a connection involves logging into a server, an application must define login parameters (such as a server user name and password) before calling vbct_connect. An application can call vbct_con_props_num and vbct_con_props_str to define login parameters.

A connection can be either synchronous or asynchronous. The CT-Library property CS_NETIO determines whether a connection will be synchronous or asynchronous.

For more information on asynchronous connections, refer to Asychronous Programming.

The maximum number of open connections per context is determined by the CS_MAX_CONNECT property (set by vbct_config). If not explicitly set, the maximum number of connections defaults to a platform-specific value.

When a connection attempt is made between a client and a server, there are two ways in which the process can fail (assuming that the system is correctly configured):

1. The machine that the server is supposed to be on is running correctly and the network is running correctly.

In this case, if there is no server listening on the specified port, the machine that the server is supposed to be on will signal the client, via a network error, that the connection cannot be formed. Regardless of the login timeout value, the connection will fail.

2. The machine that the server is on is down.

In this case, the machine that the server is supposed to be on will not respond. Because "no response" is not considered to be an error, the network will not signal the client that an error has occurred. However, if a login timeout period has been set, a timeout error will occur when the client fails to receive a response within the set period.

To close a connection, an application calls vbct_close.

See Also

vbct_close, vbct_con_alloc, vbct_con_drop, vbct_con_props_num, vbct_con_props_str, vbct_remote_pwd

vbct_data

This function is used to retrieve the data from a column for the current row within the results set.   This function is unique to SQL-Sombrero/VBX.

Syntax

vbct_data(command&, column&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

column&

The column number within a row of data returned.

Results

vbct_data returns a string representation of the data in the requested column for the last row retrieved by using the vbct_fetch function.

Remarks

See Also

vbct_do_binds, vbct_fetch

vbct_data_info

Define or retrieve a data I/O descriptor structure.

Syntax

vbct_data_info(command&, action&, colnum&, iodesc&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

action&

One of the following symbolic values:

Value of action:	**vbct_data_info:**
CS_SET	Defines an I/O descriptor.
CS_GET	Retrieves an I/O descriptor.

colnum&

The number of the text or image column whose I/O descriptor is being retrieved.

If action is CS_SET, pass colnum as CS_UNUSED.

If action is CS_GET, colnum refers to the select-list id of the text or image column. The first column in a select statement's select-list is column number 1, the second number 2, and so forth. An application must select a text or image column before it can update the column.

colnum must represent a text or image column.

iodesc&

A pointer to a CS_IODESC variable. A CS_IODESC variable contains information describing text or image data.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

vbct_data_info defines or retrieves a CS_IODESC, also called an "I/O descriptor structure" for a text or image column.

An application calls vbct_data_info to retrieve an I/O descriptor after calling vbct_get_data to retrieve a text or image column value that it plans to update at a later time. This I/O descriptor contains the text pointer and text timestamp that the server uses to manage updates to text or image columns.

After retrieving an I/O descriptor, a typical application changes only the values of the locale, total_txtlen, and log_on_update fields before using the I/O descriptor in an update operation:

- The total_txtlen field of the CS_IODESC represents the total length, in bytes, of the new text or image value.

- The log_on_update field in the CS_IODESC to indicate whether or not the server should log the update.

- The locale field of the CS_IODESC points to a CS_LOCALE structure containing localization information for the value, if any.

An application calls vbct_data_info to define an I/O descriptor before calling vbct_send_data to send a chunk or image data to the server. Both of these calls occur during a text or image update operation.

A successful text or image update generates a parameter result set that contains the new text timestamp for the text or image value. If an application plans to update the text or image value a second time, it must save this new text timestamp and copy it into the CS_IODESC for the value before calling vbct_data_info to define the CS_IODESC for the update operation.

It is illegal to call vbct_data_info to retrieve the I/O descriptor for a column before calling vbct_get_data for the column.

However, this vbct_get_data call does not have to actually retrieve any data. This means an application can call vbct_get_data with a buflen of 0, and then call vbct_data_info to retrieve the descriptor. This technique is useful when an application needs to determine the length of a text or image value before retrieving it

See Also

vbct_get_data, vbct_send_data

vbct_deinstall_callbacks

De-installs a CT-Library callback routine.

Syntax

vbct_deinstall_callbacks(context&, type&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

type&

The type of callback routine of interest. The following table lists the symbolic values that are legal for type.

Value of type:	**vbct_deinstall_callback installs:**
CS_CLIENTMSG_CB	A client message callback.
CS_COMPLETION_CB	A completion callback.
CS_ENCRYPT_CB	An encryption callback.
CS_MESSAGE_CB	A message callback.
CS_CHALLENGE_CB	A negotiation callback.
CS_SERVERMSG_CB	A server message callback.
CS_NOTIF_CB	A registered procedure notification callback.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection Refer to Asychronous Programming.

Remarks

To de-install an existing callback routine, an application can call vbct_deinstall_callback with func as NULL. An application can also install a new callback routine at any time. The new callback will automatically replace any existing callback.

See Also

vbct_install_callbacks

vbct_describe

Return a description of result data.

Syntax

vbct_describe(command&, item&, datafmt(CS_DATAFMT))

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

item&

An integer representing the result item of interest.

When retrieving a column description, item is the column's column number. The first column in a select statement's select-list is column number 1, the second number 2, and so forth.

When retrieving a compute column description, item is the column number of the compute column. Compute columns are returned in the order in which they are listed in the compute clause. The first column returned is number 1.

When retrieving a return parameter description, item is the parameter number of the parameter. The first parameter returned by a stored procedure is number 1. Stored procedure return parameters are returned in the same order as the parameters were originally specified in the stored procedure's create procedure statement. This is not necessarily the same order as specified in the RPC command that invoked the stored procedure. In determining what number to pass as item do not count non-return parameters. For example, if the second parameter in a stored procedure is the only return parameter, pass item as 1.

When retrieving a stored procedure return status description, item must be 1, as there can be only a single status in a return status result set.

When retrieving format information, item takes a column or compute column number.

NOTE: An application cannot call vbct_describe after vbct_results indicates a result set of type CS_MSG_RESULT. This is because a result type of CS_MSG_RESULT has no data items associated with it. Parameters associated with a message are returned as a CS_PARAM_RESULT result set.

Likewise, an application cannot call vbct_describe after vbct_results sets its result_type parameter to CS_CMD_DONE, CS_CMD_SUCCEED, or CS_CMD_FAIL to indicate command status information.

datafmt(CS_DATAFMT)

A pointer to a CS_DATAFMT structure. vbct_describe fills datafmt with a description of the result data item referenced by item.

vbct_describe fills in the following fields in the CS_DATAFMT:

Field name:	**For which types of result items?**	**vbct_describe sets the field to:**
name	Regular columns, column formats, and return parameters.	The null-terminated name of the data item, if any. A NULL name is indicated by a

		namelen of 0.
namelen	Regular columns, column formats, and return parameters.	The actual length of the name, not including the null terminator. 0 to indicate a NULL name.
datatype	Regular columns, column formats, return parameters, return status, compute columns, and compute column formats.	A type constant (CS_xxx_TYPE) representing the datatype of the item. Refer to Types. All type constants listed on the Types topics page are valid, with the exceptions of CS_VARCHAR_TYPE and CS_VARBINARY_TYPE. A return status has a datatype of CS_INT_TYPE. A compute column's datatype depends on the type of the underlying column and the aggregate operator that created the column.
format	Not used.	
maxlength	Regular columns, column formats, and return parameters.	The maximum possible length of the data for the column or parameter.
scale	Regular columns, column formats, return parameters, compute columns, or compute column formats of type numeric or decimal.	The scale of the result data item.
precision	Regular columns, column formats, return parameters, compute columns, or compute column formats of type numeric or decimal.	The precision of the result data item.
status	Regular columns and column formats.	A bitmask of the following symbols, or-ed together: CS_CANBENULL to indicate that the column can contain NULL values. CS_HIDDEN to indicate that the column is a "hidden" column that has been exposed. CS_IDENTITY to indicate that the column is an identity column. CS_KEY to indicate the column is part of the key for a table. CS_VERSION_KEY to indicate the column is part of the version key for the row. CS_TIMESTAMP to indicate the column is a timestamp column. CS_UPDATABLE to indicate that the column is an updatable cursor column.
count	Regular columns, column formats, return parameters, return status, compute columns, and compute column formats.	count represents the number of rows copied to program variables per vbct_fetch call.vbct_describe sets count to 1 to provide a default value in case an application uses vbct_describe's return CS_DATAFMT as vbct_do_bind's input CS_DATAFMT.
usertype	Regular columns, column formats, and return parameters.	The SQL Server user-defined datatype of the column or parameter, if any. usertype is set in addition to (not instead of) datatype.
locale	Regular columns, column formats, return parameters, return status, compute columns, and compute column formats.	A pointer to a CS_LOCALE structure that contains locale information for the data. This pointer can be NULL.

Results

Returns:	**To Indicate:**

CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. vbct_describe returns CS_FAIL if item does not represent a valid result data item.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer Asychronous Programming.

Remarks

An application can use vbct_describe to retrieve a description of a regular result column, a return parameter, a stored procedure return status number, or a compute column.

An application can also use vbct_describe to retrieve format information. CT-Library indicates that format information is available by setting vbct_results' result_type to CS_ROWFMT_RESULT or CS_COMPUTEFMT_RESULT.

An application cannot call vbct_describe after vbct_results sets its result_type parameter to CS_MSG_RESULT, CS_CMD_SUCCEED, CS_CMD_DONE, or CS_CMD_FAIL because there are no result items to describe.

An application can call vbct_res_info to find out how many result items are present in the current result set.

An application generally needs to call vbct_describe to describe a result data item before it binds the result item to a program variable using vbct_do_binds.

See Also

vbct_fetch, vbct_res_info, vbct_results

vbct_diag

Manage in-line error handling.

Syntax

vbct_diag(connection&, operation&, type&, index&, buffer(Any))

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

operation&

The operation to perform. The table in the Summary of Parameters below lists the symbolic values that are legal for operation.

type&

Depending on the value of operation, type indicates either the type of structure to receive message information, the type of message on which to operate, or both. The following table lists the symbolic values that are legal for type:

Value of type:	**To indicate:**
SQLCA_TYPE	A SQLCA structure.
SQLCODE_TYPE	A SQLCODE structure, which is a long integer.
SQLSTATE_TYPE	A SQLSTATE structure, which is an array of bytes.
CS_CLIENTMSG_TYPE	A CS_CLIENTMSG structure. Also used to indicate CT-Library messages.
CS_SERVERMSG_TYPE	A CS_SERVERMSG structure. Also used to indicate server messages.
CS_ALLMSG_TYPE	CT-Library and server messages.

index&

The index of the message of interest. The first message has an index of 1, the second an index of 2, and so forth.

If type is CS_CLIENTMSG_TYPE, then index refers to CT-Library messages only. If type is CS_SERVERMSG_TYPE, then index refers to server messages only. If type is CS_ALLMSG_TYPE, then index refers to CT-Library and server messages combined.

buffer(Any)

A pointer to data space.

Depending on the value of operation, buffer can point to a structure or a CS_INT.

Summary of Parameters Table

Value of operation:	**vbct_diag:**	**type is:**	**index is:**	**buffer is:**
CS_INIT	Initializes in-line error handling.	CS_UNUSED	CS_UNUSED	NULL
CS_MSGLIMIT	Sets the maximum	CS_CLIENTMSG_TYP	CS_UNUSED	A pointer to an

	number of messages to store.	E to limit CT-Library messages only. CS_SERVERMSG_ TYPE to limit server messages only. CS_ALLMSG_TYPE to limit the total number of CT-Library and server messages combined.		integer value.
CS_CLEAR	Clears message information for this connection. If buffer is not NULL and type is not CS_ALLMSG_ TYPE, vbct_diag also clears the buffer structure by initializing it with blanks and/or NULLs, as appropriate.	One of the legal type values: If type is CS_CLIENTMSG_TYP E vbct_diag clears CT-Library messages only. If type is CS_SERVERMSG_ TYPE, vbct_diag clears server messages only. If type has any other legal value, vbct_diag clears both CT-Library and server messages.	CS_UNUSED	A pointer to a structure whose type is defined by type, or NULL.
CS_GET	Retrieves a specific message.	Any legal type value except CS_ALLMSG_TYPE. If type is CS_CLIENTMSG_TYP E a CT-Library message is retrieved into a CS_CLIENTMSG structure. If type is CS_SERVERMSG_ TYPE, a server message is retrieved into a CS_SERVERMSG structure. If type has any other legal value, then either a CT-Library or server message is retrieved.	The one- based index of the message to retrieve.	A pointer to a structure whose type is defined by type.
CS_STATUS	Returns the current number of stored messages.	CS_CLIENTMSG_TYP E to retrieve the number of CT-Library messages. CS_SERVERMSG_ TYPE to retrieve the number of server messages. CS_ALLMSG_TYPE to retrieve the total number of CT-Library and server messages combined.	CS_UNUSED	A pointer to an integer variable.
CS_EED_CMD	Sets buffer to the address of the CS_COMMAND structure containing extended error data.	CS_SERVERMSG_ TYPE	The one- based index of the message for which extended error data is available.	A pointer to a pointer variable.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. vbct_diag returns CS_FAIL if the original error has made the connection unusable.
CS_NOMSG	The application attempted to retrieve a message whose index is higher than the highest valid index. For example, the application attempted to retrieve message number 3, when there are only 2 messages queued.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Common reasons for a vbct_diag failure include:

- Invalid connection.
- Inability to allocate memory.
- Invalid parameter combination.

Remarks

A CT-Library application can handle CT-Library and server messages in the following two ways:

- The application can call vbct_install_callbacks to install client message and server message callbacks to handle CT-Library and server messages.
- The application can handle CT-Library and server messages in-line, using vbct_diag.

It is possible for an application to switch back and forth between the two methods. Refer to Error and Message Handling.

vbct_diag manages in-line message handling for a specific connection. If an application has more than one connection, it must make separate vbct_diag calls for each connection.

An application cannot use vbct_diag at the context level. This means an application cannot use vbct_diag to retrieve messages generated by routines that take a CS_CONTEXT (and no CS_CONNECTION) as a parameter. These messages are unavailable to an application that is using in-line error handling.

An application can perform operations on either CT-Library messages, server messages, or both.

For example, an application can clear CT-Library messages without affecting server messages:

```
vbct_diag(connection, CS_CLEAR, CS_CLIENTMSG,
        CS_UNUSED, NULL);
```

vbct_diag allows an application to retrieve message information into standard CT-Library structures (CS_CLIENTMSG and CS_SERVERMSG) or a SQLCA, SQLCODE, or SQLSTATE. When retrieving messages, vbct_diag assumes that buffer points to a structure of the type indicated by type.

An application that is retrieving messages into a SQLCA, SQLCODE, or SQLSTATE must set the CT-Library property CS_EXTRA_INF to CS_TRUE. This is because the SQL structures require information that is not ordinarily returned by CT-Library's error handling mechanism.

An application that is not using the SQL structures can also set CS_EXTRA_INF to CS_TRUE. In this case, the extra information is returned as standard CT-Library messages.

If vbct_diag does not have sufficient internal storage space in which to save a new message, it throws away all unread messages and stops saving messages. The next time it is called with operation as CS_GET, it returns a special message to indicate the space problem.

After returning this message, vbct_diag starts saving messages again.

Initializing In-Line Error Handling

To initialize in-line error handling, an application calls vbct_diag with operation as CS_INIT.

Generally, if a connection will use in-line error handling, an application should call vbct_diag to initialize in-line error handling for a connection immediately after allocating it.

Clearing Messages

To clear message information for a connection, an application calls vbct_diag with operation as CS_CLEAR.

To clear CT-Library messages only, an application passes type as CS_CLIENTMSG_TYPE.

To clear server messages only, an application passes type as CS_SERVERMSG.

To clear both CT-Library and server messages, pass type as SQLCA, SQLCODE, or CS_ALLMSG_TYPE.

If type is not CS_ALLMSG_TYPE:

vbct_diag assumes that buffer points to a structure of type type.

vbct_diag clears the buffer structure by setting it to blanks and/or NULLs, as appropriate.

Message information is not cleared until an application explicitly calls vbct_diag with operations as CS_CLEAR. Retrieving a message does not remove it from the message queue.

Retrieving Messages

To retrieve message information, an application calls vbct_diag with operation as CS_GET, type as the type of structure in which to retrieve the message, index as the one-based index of the message of interest, and buffer as a structure of the appropriate type.

If type is CS_CLIENTMSG_TYPE, then index refers only to CT-Library messages. If type is CS_SERVERMSG_TYPE, index refers only to server messages. If type has any other value, index refers to the collective "queue" of both types of messages combined.

vbct_diag fills in the buffer structure with the message information.

If an application attempts to retrieve a message whose index is higher than the highest valid index, vbct_diag returns CS_NOMSG to indicate that no message is available.

See the SQLCA, SQLCODE, CS_CLIENTMSG and CS_SERVERMSG topics pages for information on these structures.

Limiting Messages

Applications running on platforms with limited memory may want to limit the number of messages that CT-Library saves.

An application can limit the number of saved CT-Library messages, the number of saved server messages, and the total number of saved messages.

To limit the number of saved messages, an application calls vbct_diag with operation as CS_MSGLIMIT and type as CS_CLIENTMSG_TYPE, CS_SERVERMSG_TYPE, or CS_ALLMSG_TYPE:

If type is CS_CLIENTMSG_TYPE, then the number of CT-Library messages is limited.

If type is CS_SERVERMSG_TYPE, then the number of server messages is limited.

If type is CS_ALLMSG_TYPE, then the total number of CT-Library and server messages combined is limited.

When a specific message limit is reached, CT-Library discards any new messages of that type. When a combined message limit is reached, CT-Library discards any new messages. If CT-Library discards messages, it saves a message to this effect.

An application cannot set a message limit that is less than the number of messages currently saved.

CT-Library's default behavior is to save an unlimited number of messages. An application can restore this default behavior by setting a message limit of CS_NO_LIMIT.

Retrieving the Number of Messages

To retrieve the number of current messages, an application calls vbct_diag with operation as CS_STATUS and type as the type of message of interest.

Getting the CS_COMMAND for Extended Error Data

To retrieve a pointer to the CS_COMMAND structure containing extended error data (if any), call vbct_diag with operation as CS_EED_CMD and type as CS_SERVERMSG_TYPE. vbct_diag sets buffer to the address of the CS_COMMAND structure containing the extended error data.

When an application retrieves a server message into a CS_SERVERMSG structure, CT-Library indicates that extended error data is available for the message by setting the CS_HASEED bit in the status field in the CS_SERVERMSG structure.

It is an error to call vbct_diag with operation as CS_EED_CMD when extended error data is not available.

Sequenced Messages and vbct_diag

If an application is using sequenced error messages, vbct_diag acts on message chunks instead of messages. This has the following effects:

- A vbct_diag(CS_GET) call with index i returns the i'th message chunk, not the i'th message.
- A vbct_diag(CS_MSGLIMIT) call limits the number of chunks, not the number of messages, that CT-Library will store.
- A vbct_diag(CS_STATUS) call returns the number of currently-stored chunks, not the number of currently-stored messages.

See Also

Error and Message Handling, vbct_install_callbacks, vbct_options_num, vbct_options_str

vbct_do_binds

Binds server results to internal SQL-Sombrero/VBX internal program variables.

Syntax

vbct_do_binds(command&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Remarks

vbct_do_binds is the equivalent of performing a ct_bind for each column in a result set. Each column is bound to an internal variable. The information is made available through a call to the vbct_data function.

vbct_do_binds should only be called after function vbct_results returns a CS_SUCCEED.

Subsequent calls to vbct_fetch will result in the data from each row being placed in the internal variables.

Binding remains in effect until vbct_results returns CS_CMD_DONE.

See Also

vbct_data, vbct_fetch, vbct_results

vbct_dynamic

Initiate a prepared dynamic SQL-statement command.

Syntax

vbct_dynamic(command&, type&, id$, buffer$)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

type&

The type of dynamic SQL command to initiate.

Value of type:	**vbct_dynamic:**	**id is:**	**buffer is:**
CS_CURSOR_DECLARE	Declares a cursor on a previously-prepared SQL statement.	The prepared statement identifier.	The cursor name.
CS_DEALLOC	De-allocates a prepared SQL statement.	The prepared statement identifier.	NULL
CS_DESCRIBE_INPUT	Retrieves input parameter information. An application can access this information via vbct_describe or via vbct_dyndesc.	The prepared statement identifier.	NULL
CS_DESCRIBE_OUTPUT	Retrieves column list information. An application can access this information via vbct_describe or via vbct_dyndesc.	The prepared statement identifier.	NULL
CS_EXECUTE	Executes a prepared SQL statement that requires zero or more parameters.	The prepared statement identifier.	NULL
CS_EXEC_IMMEDIATE	Execute a literal SQL statement.	NULL.	The SQL statement to execute.
CS_PREPARE	Prepares a SQL statement.	The prepared statement identifier.	The SQL statement to prepare.

id$

A pointer to the statement identifier. This identifier is defined by the application and must conform to server standards.

buffer$

A pointer to data space.

Results

Returns:	**To Indicate:**

CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Initiating a command is the first step in sending it to a server.

Follow these steps to send a command to a server:

- Initiate the command by calling vbct_dynamic. This routine sets up internal structures that are used in building a command stream to send to the server.
- Pass parameters for the command, if required. Most applications will pass parameters by calling vbct_param once for each parameter that the command requires, but it is also possible to pass parameters for a command by using vbct_dyndesc.
- Send the command to the server by calling vbct_send.
- Verify the success of the command by calling vbct_results.

This last step does not imply that an application need only call vbct_results once. If the value of vbct_results' result_type parameter indicates that there are fetchable results, the application will most likely process the results using a loop controlled by vbct_results. See the Open Client Client-Library/C Programmer's Guide for a discussion of processing results.

About Prepared Statements

A prepared SQL statement is a SQL statement which is compiled and stored by a server. Each prepared statement is associated with a unique identifier.

An application can prepare an unlimited number of statements, but identifiers for prepared statements must be unique within a connection.

Although the command structure used to prepare a statement can be different from the one used to execute it, both of the command structures must belong to the same connection.

If a prepared statement is a Transact-SQL command containing host variables, each variable must begin with a colon (:).

If a prepared statement requires parameters, they are passed using vbct_param or vbct_dyndesc at execute time.

Once a statement is successfully prepared, it can be executed repeatedly until it is de-allocated.

Preparing a Statement

To initiate a command to prepare a statement, an application calls vbct_dynamic with type as CS_PREPARE.

Getting a Description of Prepared Statement Input

To initiate a command to get a description of prepared statement input parameters, an application calls vbct_dynamic with type as CS_DESCRIBE_INPUT.

An application typically retrieves a description of prepared statement input parameters before passing input values to a prepared statement.

Getting a Description of Prepared Statement Output

To initiate a command to get a description of prepared statement output columns, an application calls vbct_dynamic with type as CS_DESCRIBE_OUTPUT.

Executing a Prepared Statement

To initiate a command to execute a prepared statement, an application calls vbct_dynamic with type as CS_EXECUTE.

Executing a Literal Statement

To initiate a command to execute a literal SQL statement, an application calls vbct_dynamic with type as CS_EXEC_IMMEDIATE.

De-allocating a Prepared Statement

To initiate a command to de-allocate a prepared statement, an application calls vbct_dynamic with type as CS_DEALLOC.

See Also

vbct_dyndesc, vbct_param, vbct_send

vbct_dyndesc

Perform operations on a dynamic SQL descriptor area.

Syntax

vbct_dyndesc(command&, descriptor$, operation&, index&, datafmt(DATAFMT), buffer(Any), buflen&, copied&, indicator%)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

descriptor$

A pointer to the name of the descriptor. Descriptor names must be unique within a connection.

operation&

The descriptor operation to initiate. The following table lists the values that are legal for operation:

Value of operation:	vbct_dyndesc:
CS_ALLOC	Allocates a descriptor.
CS_DEALLOC	De-allocates a descriptor.
CS_GETATTR	Retrieves a parameter or result item's attributes.
CS_GETCNT	Retrieves the number of parameters or columns.
CS_SETATTR	Sets a parameter's attributes.
CS_SETCNT	Sets the number of parameters or columns.
CS_USE_DESC	Associates a descriptor with a statement or a command structure

index&

When used, an integer variable.

Depending on the value of operation, index can be either the zero-based index of a descriptor item or the number of items associated with a descriptor.

datafmt(DATAFMT)

When used, a pointer to a CS_DATAFMT structure.

buffer(Any)

When used, a pointer to data space.

buflen&

When used, buflen is the length, in bytes, of the buffer data.

copied&

When used, a pointer to an integer variable. vbct_dyndesc sets copied to the length, in bytes, of the data placed in buffer.

indicator%

When used, a pointer to an indicator variable.

The following table lists the possible values of indicator:

Value of operation:	Value of indicator:	To Indicate:
CS_GETATTR	-1	Truncation of a server value by CT-Library.
	0	No truncation.
	integer value	Truncation of an application value by the server.
CS_SETATTR	-1	The parameter has a null value.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_ROW_FAIL	A recoverable error occurred. Recoverable errors include conversion errors that occur while copying values to program variables as well as memory allocation failures.
CS_CANCELED	The dynamic SQL operation was canceled.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

A dynamic SQL descriptor area contains information about the input parameters to a dynamic SQL statement or the result data items generated by the execution of a dynamic SQL statement.

Although vbct_dyndesc takes a CS_COMMAND structure as a parameter, the scope of a dynamic SQL descriptor area is a CT-Library context. For example:

- Descriptor names must be unique within a context.
- An application can use any command structure within a context to reference the context's descriptor areas. For example, a descriptor area allocated through one command structure can be de-allocated by another command structure within the same context.

Allocating a Descriptor

To allocate a descriptor, an application calls vbct_dyndesc with operation as CS_ALLOC.

The following table lists parameter values for CS_ALLOC operations:

descriptor, desclen:	index:	datafmt:	buffer, buflen:	copied:	indicator :
The name of the descriptor to allocate, the length of the name or CS_NULLTERM.	The maximum number of items that the des-criptor will accom-modate.	NULL	NULL, CS_UNUSED	NULL	NULL

De-allocating a Descriptor

To de-allocate a descriptor, an application calls vbct_dyndesc with operation as CS_DEALLOC.

The following table lists parameter values for CS_DEALLOC operations:

descriptor, desclen:	index:	datafmt:	buffer, buflen:	copied:	indicator :
The name of the descriptor to de-allocate, the length of the name or CS_NULLTERM.	CS_UNUSED	NULL	NULL, CS_UNUSED	NULL	NULL

Retrieving a Parameter or Result Item's Attributes

To retrieve a parameter's or a result data item's attributes, an application calls vbct_dyndesc with operation as CS_GETATTR.

The following table lists parameter values for CS_GETATTR operations:

descriptor, desclen:	index:	datafmt:	buffer, buflen:	copied:	indicator:
The name of the descriptor of interest, the length of the name or CS_NULLTERM.	The number of the item whose description is being requested.	As an input parameter, datafmt describes buffer. vbct_dyndesc overwrites datafmt with a description of the item.	If supplied, buffer is set to the value of the item. If buffer is NULL, only a description of the item is re-turned. buflen must be CS_UNUSED. datafmt.maxlength describes buffer's length.	If supplied, copied is set to the number of bytes placed in buffer. Can be NULL.	If supplied, indicator is set to the value of the item's indicator. Can be NULL.

An application needs to set the datafmt fields for a CS_GETATTR operation exactly as they would be set for a ct_bind call. The following table lists the fields that are used:

Field name:	Set the field to:
datatype	The datatype of the buffer variable.
format	A bit-mask of format symbols.
maxlength	The length of the buffer data space.
scale	The scale of a numeric or decimal buffer; ignored for all other datatypes.
precision	The precision of a numeric or decimal buffer; ignored for all other datatypes.
count	0 or 1
locale	A pointer to a valid CS_LOCALE structure or NULL.
All other fields	Are ignored.

vbct_dyndesc(CS_GETATTR) sets the datafmt fields exactly as vbct_describe would set them. The following table lists the fields in datafmt that vbct_dyndesc sets:

Field name:	vbct_dyndesc sets the field to:
name	The null-terminated name of the data item, if any. A NULL name is indicated by a namelen of 0.
namelen	The actual length of the name, not including the null terminator. 0 to indicate a NULL name.
datatype	The datatype of the item. All datatypes listed on the types topics page are valid, with the exceptions of CS_VARCHAR and CS_VARBINARY. Refer to Types.
maxlength	The maximum possible length of the data for the column or parameter.
scale	The scale of the result data item.

precision	The precision of the result data item.
status	A bitmask of the following symbols, or-ed together: CS_CANBENULL to indicate that the column can contain NULL values. CS_HIDDEN to indicate that the column is a "hidden" column that has been exposed. CS_IDENTITY to indicate that the column is an identity column. CS_KEY to indicate the column is part of the key for a table. CS_VERSION_KEY to indicate the column is part of the version key for the row. CS_TIMESTAMP to indicate the column is a timestamp column. CS_UPDATABLE to indicate that the column is an updatable cursor column.
count	count represents the number of rows copied to program variables per vbct_fetch call.vbct_dyndesc sets count to 1 to provide a default value in case an application uses vbct_dyndesc's return CS_DATAFMT as ct_do_binds' input CS_DATAFMT.
usertype	The SQL Server user-defined datatype of the column or parameter, if any. usertype is set in addition to (not instead of) datatype.
locale	A pointer to a CS_LOCALE structure that contains locale information for the data. This pointer can be NULL.

Retrieving the Number of Parameters or Columns

To retrieve the number of parameters or result items a descriptor can describe, an application calls vbct_dyndesc with operation as CS_GETCNT.

vbct_dyndesc sets buffer to the number of dynamic parameter specifications or the number of columns in the dynamic SQL statement's select list, depending on whether input parameters or output columns are being described.

The following table lists parameter values for CS_GETCNT operations:

descriptor, desclen:	**index:**	**datafmt:**	**buffer, buflen:**	**copied:**	**indicator:**
The name of the descriptor of interest, the length of the name or CS_NULLTERM.	CS_UNUSED	NULL	A pointer to a CS_INT, CS_UNUSE D	If supplied, copied is set to the number of bytes placed in buffer. Can be NULL.	NULL

Setting a Parameter's Attributes

To set a parameter's attributes, an application calls vbct_dyndesc with operation as CS_SETATTR.

The following table lists parameter values for CS_SETATTR operations:

descriptor, desclen:	**index:**	**datafmt:**	**buffer, buflen:**	**copied:**	**indicator:**
The name of the descriptor of interest, the length of the name or CS_NULLTERM.	The number of the item whose description is being set.	datafmt contains a description of the item.	A pointer to the value of the item, the length of the value. Pass buflen as CS_UNUSED if buffer points to a fixed-length type.	NULL	If supplied, indicator is the value of the item's indicator. If indicator is -1 then buffer is ignored and the value of the item is set to NULL.

indicator
can be
NULL

An application needs to set the datafmt fields for a CS_SETATTR operation exactly as they would be set for a vbct_param call. The following table lists the fields that are used:

Field name:	**Set the field to:**
name	The name of the parameter.
namelen	The length of the name or CS_NULLTERM.
datatype	The datatype of the item being set.
maxlength	For variable-length return parameters, maxlength is the maximum number of bytes to be returned for this parameter.maxlength is ignored if status is CS_INPUTVALUE or if datatype represents a fixed-length type.
status	CS_INPUTVALUE, CS_UPDATECOL, or CS_RETURN. CS_UPDATECOL indicates an update column for a cursor declare command.CS_RETURN indicates a return parameter.
locale	A pointer to a valid CS_LOCALE structure or NULL.
All other fields	Are ignored.

Setting the Number of Parameters or Columns

To set the number of parameters or columns a descriptor can describe, an application calls vbct_dyndesc with operation as CS_SETCNT.

The following table lists parameter values for CS_SETCNT operations:

descriptor, desclen:	**index:**	**datafmt:**	**buffer, buflen:**	**copied:**	**indicator :**
The name of the descriptor to allocate, the length of the name or CS_NULLTERM.	The new descriptor count.	NULL	NULL, CS_UNUSED	NULL	NULL

Associating a Descriptor with a Statement or Command Structure

To associate a descriptor with a prepared statement or command structure, an application calls vbct_dyndesc with operation as CS_USE_DESC.

The following table lists parameter values for CS_USE_DESC operations:

descriptor, desclen:	**index:**	**datafmt:**	**buffer, buflen:**	**copied:**	**indicator:**
The name of the descriptor to allocate, the length of the name or CS_NULLTERM.	CS_UNUSED	NULL	NULL, CS_UNUSED	NULL	NULL

See Also

vbct_dynamic, vbct_fetch

vbct_exit

Exit CT-Library.

Syntax

vbct_exit(context&, option&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

option&

vbct_exit can behave in different ways, depending on the value specified for option. The following table lists the symbolic values that are legal for option:

Value of option:	Behavior of vbct_exit:
CS_UNUSED	vbct_exit closes all open connections for which no results are pending and terminates CT-Library for this context. If results are pending on one or more connections, vbct_exit returns CS_FAIL and does not terminate CT-Library.
CS_FORCE_EXIT	vbct_exit closes all open connections for this context, whether or not any results are pending, and terminates CT-Library for this context.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Remarks

vbct_exit terminates CT-Library for a specific context. It closes all open connections, de-allocates internal data space and cleans up any platform-specific initialization.

vbct_exit must be the last CT-Library routine called within an CT-Library context.

If an application finds it needs to call CT-Library routines after it has called vbct_exit , it can re-initialize CT-Library by calling vbct_init again.

If results are pending on any of the context's connections and option is not passed as CS_FORCE_EXIT, vbct_exit returns CS_FAIL. This means that CT-Library is not correctly terminated and that the application must call vbct_exit again after handling the connections' pending results.

vbct_exit always completes synchronously, even if asynchronous network I/O has been specified for any of the context's connections.

An application can call vbct_close to close a single connection.

If vbct_init is called for a context, it is an error to de-allocate the context before calling vbct_exit

See Also

vbct_close , vbct_init

vbct_fetch

Fetch result data.

Syntax

vbct_fetch(command&, type&, offset&, option&, rows_read&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

type&

This parameter is currently unused and must be passed as CS_UNUSED in order to ensure compatibility with future versions of CT-Library.

offset&

This parameter is currently unused and must be passed as CS_UNUSED in order to ensure compatibility with future versions of CT-Library.

option&

This parameter is currently unused and must be passed as CS_UNUSED in order to ensure compatibility with future versions of CT-Library.

rows_read&

A pointer to an integer variable. vbct_fetch sets rows_read to the number of rows read by the vbct_fetch call.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully vbct_fetch places the number of rows read in rows_read. The application must continue to call vbct_fetch, as the result data is not yet completely fetched.
CS_END_DATA	All rows of the current result set have been fetched. The application should call vbct_results to get the next result set.
CS_ROW_FAIL	A recoverable error occurred while fetching a row. Recoverable errors include memory allocation failures and conversion errors that occur while copying row values to program variables. An application can continue calling vbct_fetch to keep retrieving rows, or can call vbct_cancel to cancel the remaining results. vbct_fetch places the number of rows fetched in rows_read. This number includes the row on which the error occurred. The application must continue to call vbct_fetch, as the result data is not yet completely fetched.
CS_FAIL	The routine failed. vbct_fetch places the number of rows fetched in rows_read. This number includes the failed row. Unless the routine failed due to application error (for example, bad parameters), additional result rows are not available. If

	vbct_fetch returns CS_FAIL, an application must call vbct_cancel with type as CS_CANCEL_ALL before using the affected command structure to send another command. If vbct_cancel returns CS_FAIL, the application must call vbcs_close(CS_FORCE_CLOSE) to force the connection closed.
CS_CANCELED	The current result set and any additional result sets have been canceled. Data is no longer available. vbct_fetch places the number of rows fetched before the cancel occurred in rows_read.
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer Asychronous Programming.

"Result data" is an umbrella term for all the types of data that a server can return to an application. These types of data include:

- Regular rows.
- Cursor rows.
- Return parameters. Types of data that are returned as parameters include message parameters, stored procedure return parameters, extended error data, and registered procedure notification parameters.
- Stored procedure status numbers.
- Compute rows.

vbct_fetch is used to fetch all of these types of data.

Conceptually, result data is returned to an application in the form of one or more rows that make up a "result set".

Regular row and cursor row result sets can contain more than one row. For example, a regular row result set might contain a hundred rows.

If array binding has been specified for the data items in a regular row or cursor row result set, then multiple rows can be fetched with a single call to vbct_fetch.

Return parameter, status number, and compute row result sets, however, only contain a single "row." For this reason, even if array binding is specified, only a single row of data is fetched.

vbct_results sets result_type to indicate the type of result available. vbct_results must indicate a result type of CS_ROW_RESULT, CS_CURSOR_RESULT, CS_PARAM_RESULT, CS_STATUS_RESULT, or CS_COMPUTE_RESULT before an application calls vbct_fetch.

After calling vbct_results, an application can:

- Process the result set by binding the result items and fetching the data. (A typical application will call vbct_describe to get data descriptions, vbct_do_binds to bind result items, vbct_fetch to fetch result rows, and vbct_data, if the result set contains large text or image values. However, an application can also use vbct_fetch and vbct_dyndesc to process a result set.)
- Discard the result set, using vbct_cancel.

If an application does not cancel a result set, it must completely process the result set by calling vbct_fetch as long as vbct_fetch continues to indicate that rows are available.

The simplest way to do this is in a loop that terminates when vbct_fetch fails to return either CS_SUCCEED or CS_ROW_FAIL. After the loop terminates, an application can use a switch-type statement against vbct_fetch's final return code to find out what caused the termination.

If a result set contains zero rows, an application's first vbct_fetch call will return CS_END_DATA.

NOTE: An application must call vbct_fetch in a loop even if a result set contains only a single row. An application must call vbct_fetch until it fails to return either CS_SUCCEED or CS_ROW_FAIL.

If a conversion error occurs when retrieving a result item, the rest of the items in the row are retrieved.

vbct_fetch returns CS_ROW_FAIL if a conversion or truncation error occurs.

Fetching Regular Rows

Regular rows can be fetched one row at a time, or several rows at once.

Fetching Return Parameters

Several types of data can be returned to an application as a parameter result set, including the following:

- Stored procedure return parameters
- Message parameters

Extended error data and registered procedure notification parameters are also returned as parameter result sets, but since an application does not call vbct_results to process these types of data, the application never sees a result type of CS_PARAM_RESULT. Instead, the row of parameters is simply available to be fetched after the application retrieves the CS_COMMAND structure containing the data.

A return parameter result set consists of a single row with a number of columns equal to the number of return parameters.

Fetching a Return Status

A stored procedure return status result set consists of a single row with a single column, the status number.

Fetching Compute Rows

Compute rows result from the compute clause of a select statement.

A compute row result set consists of a single row with a number of columns equal to the number of aggregate operators in the compute clause that generated the row.

Each compute row is considered to be a distinct result set.

See Also

vbct_describe, vbct_get_data, vbct_results.

vbct_get_data

Read a chunk of data from the server.

Syntax

vbct_get_data(command&, item&, buffer(Any), buflen&, outlen&)

Parameters

command&

The parameter passed is the command pointer which is passed from the vbct_cmd_alloc function.

item&

An integer representing the data item of interest.   When using vbct_get_data to retrieve data for more than one item in a result set, item can be incremented only.   This means an application cannot retrieve data for item number 3 after it has retrieved data for item number 4.

When retrieving a column, item is the column's column number.   The first column in a select statement's select-list is column number 1, the second number 2, and so forth.

When retrieving a compute column, item is the column number of the compute column.   Compute columns are returned in the order in which they are listed in the compute clause.   The first column returned is number 1.

When retrieving a return parameter, item is the parameter number of the parameter.   The first parameter returned by a stored procedure is number 1.   Stored procedure return parameters are returned in the same order as the parameters were originally specified in the stored procedure's create procedure statement.   This is not necessarily the same order as specified in the RPC command that invoked the stored procedure.   In determining what number to pass as item do not count non-return parameters. For example, if the second parameter in a stored procedure is the only return parameter, pass item as 1.

When retrieving a stored procedure return status, item must be 1, as there can be only a single status in a return status result set.

buffer(Any)

A pointer to data space. vbct_get_data fills buffer with a buflen-sized chunk of the column's value.

buffer cannot be NULL.

buflen&

The length, in bytes, of buffer.

If buflen is 0, vbct_get_data updates the I/O descriptor for the item without retrieving any data.

buflen is required even for fixed-length buffers, and cannot be CS_UNUSED.

outlen&

A poiner to an integer variable.

If outlen is supplied, vbct_get_data sets outlen to the number of bytes placed in buffer.

Results

Returns:	**To Indicate:**
CS_SUCCEED	vbct_get_data successfully retrieved a chunk of data that is not the last chunk of data for this column.
CS_FAIL	The routine failed. Unless the routine failed due to application error(for example, bad parameters), additional result data is not available.
CS_END_ITEM	vbct_get_data successfully retrieved the last chunk of data for this column. This is not the last column in the row.
CS_END_DATA	vbct_get_data successfully retrieved the last chunk of data for this column. This is the last column in the row.
CS_CANCELED	The operation was canceled. Data for this result set is no longer available.
CS_PENDING	Asynchronous network I/O is in effect. Refer to Asychronous Programmingfor more information.

Remarks

An application typically calls vbct_get_data in a loop to retrieve large text or image values, although it can be used on columns of any datatype. Each call to vbct_get_data retrieves a buflen-sized chunk of data.

vbct_get_data retrieves data exactly as it is sent by the server. No conversion is performed. For this reason, care must be taken when interpreting data contained in buffer. In particular, CS_CHAR data may not be null-terminated and multi-byte character strings may be broken within a byte sequence defining a single character.

Only those columns following the last bound column are available to vbct_get_data. Data in unbound columns that precede bound columns is discarded. For example, if an application selects columns number 1 through 4 and binds columns number 1 and 3, the application cannot use vbct_get_data to retrieve the data for column 2, but can use vbct_get_data to retrieve the data for column 4.

Once data has been retrieved for a column, it is no longer available.

If an application reads a text or image column that it will need to update at a later time, it needs to retrieve an I/O descriptor for the column. To do this, an application can call vbct_data_info after calling vbct_get_data for the column.

NOTE: An application cannot retrieve an I/O descriptor for a column before it has called vbct_get_data for the column. However, this vbct_get_data call does not have to actually retrieve any data. An application can call vbct_get_data with a buflen of 0, and then call vbct_data_info to retrieve the descriptor. This technique is useful when an application needs to determine the length of a text or image value before retrieving it.

See Also

vbct_data_info, vbct_fetch, vbct_send_data

vbct_getformat

Return the server user-defined format string associated with a result column.

Syntax

vbct_getformat (command&, colnum&, buffer$, outlen&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

colnum&

The number of the column whose user-defined format is desired. The first column in a select statement's select-list is column number 1, the second number 2, and so forth.

buffer$

A pointer to the space in which vbct_getformat will place a null-terminated format string.

outlen&

A pointer to an integer variable.

If outlen is supplied, vbct_getformat sets outlen to the length, in bytes, of the format string. This length includes the null terminator.

If the format string is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the string.

If no format string is associated with the column identified by colnum, vbct_getformat sets outlen to 1 (for the null terminator).

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

An application can call vbct_getformat after vbct_results indicates results of type CS_ROW_RESULT.

If no format string is associated with the column identified by colnum, vbct_getformat sets outlen to 1.

Typical applications will not use vbct_getformat, which is provided primarily for gateway applications support.

See Also

vbct_describe

vbct_getloginfo

Transfer TDS login response information from a CS_CONNECTION structure to a newly-allocated CS_LOGINFO structure.

Syntax

vbct_getloginfo (connection&, logptr&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

logptr&

A pointer to a program variable which vbct_getloginfo sets to the address of a newly-allocated CS_LOGINFO structure.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

TDS (Tabular Data Stream) is a communications protocol used for the transfer of requests and request results between clients and servers.

There are two reasons an application might call vbct_getloginfo:

- If it is an Open Server gateway application using TDS pass-through.
- In order to copy login properties from an open connection to a newly-allocated connection structure.

NOTE: Do not call vbct_getloginfo from within a completion callback routine. vbct_getloginfo calls system-level memory functions which may not be re-entrant.

TDS Pass-Through

When a client connects directly to a server, the two programs negotiate the TDS format they will use to send and receive data. When a gateway application uses TDS pass-through, the gateway forwards TDS packets between the client and a remote server without examining or processing them. This means the remote server and the client must agree on a TDS format to use.

vbct_getloginfo is the third of four calls, two of them Server Library calls, that allow a client and a remote server to negotiate a TDS format. The calls, that can only be made in an Open Server SRV_CONNECT event handler, are as follows:

- srv_getloginfo to allocate a CS_LOGINFO structure and fill it with TDS information from a client login request.
- vbct_setloginfo to transfer the TDS information retrieved in step 1 from the CS_LOGINFO structure to a CT-Library CS_CONNECTION structure.   The gateway uses this CS_CONNECTION structure in the vbct_connect call which establishes its connection with the remote server.
- vbct_getloginfo to transfer the remote server's response to the client's TDS information from the CS_CONNECTION structure into a newly-allocated CS_LOGINFO structure.
- srv_setloginfo to send the remote server's response, retrieved in step 3, to the client.

See Also

vbct_recvpassthru, vbct_sendpassthru, vbct_setloginfo

vbct_init

Initialize CT-Library for an application context.

Syntax

vbct_init(context&, version&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

version&

The version of CT-Library behavior that the application expects. The following table lists the symbolic values that are legal for version:

Value of version:	To Indicate:	Features Supported:
CS_VERSION_100	10.0 behavior.	Cursors, registered procedures, remote procedure calls. This is the initial version of CT-Library.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_MEM_ERROR	The routine failed due to a memory allocation error.
CS_FAIL	The routine failed for other reasons. vbct_init returns CS_FAIL if CT-Library cannot provide version-level behavior.

A vbct_init failure does not typically make context unusable. Instead of dropping the context structure, an application can try calling vbct_init again with the same context pointer.

Remarks

vbct_init sets up internal control structures and defines the version of CT-Library behavior that the application expects.

vbct_init must be the first CT-Library routine called in a CT-Library application context. Other CT-Library routines will fail if they are called before vbct_init.

NOTE: A CT-Library application can call CT-Library routines before calling vbct_init (but must call the CT-Library routine vbcs_ctx_alloc before calling vbct_init).

If vbct_init returns CS_SUCCEED, CT-Library will provide the requested behavior, regardless of the actual version of CT-Library in use. If CT-Library cannot provide the requested behavior, vbct_init returns CS_FAIL. Generally speaking, higher-level versions of CT-Library can provide lower-level behavior, but lower versions cannot provide higher-level behavior.

Because an application calls vbct_init before it sets up error handling, an application must check vbct_init's return code in order to detect failure.

It is not an error for an application to call vbct_init multiple times for the same context. If this occurs, only the first call has any effect. CT-Library provides this functionality because some applications cannot guarantee which of several modules will execute first. If this happens, each module needs to contain a call to vbct_init.

version is the version of CT-Library behavior that the application expects. version determines the value of the context's CS_VERSION property. Connections allocated within a context use default CS_TDS_VERSION values based on their parent context's CS_VERSION level.

See Also

vbcs_ctx_alloc, vbct_exit

vbct_install_callbacks

Installs a CT-Library callack routine.

Syntax

vbct_install_callbacks(context&, type&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

type&

The type of callback routine of interest. The folloing table lists the symbolic values that are legal for type.

Value of type:	**vbct_install_callbacks installs:**
CS_CLIENTMSG_CB	A client message callback.
CS_COMPLETION_CB	A completion callback.
CS_ENCRYPT_CB	An encryption callback.
CS_MESSAGE_CB	A message callback.
CS_CHALLENGE_CB	A negotiation callback.
CS_SERVERMSG_CB	A server message callback.
CS_NOTIF_CB	A registered procedure notification callback.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

A typical application will use vbct_install_callbacks only to install callback routines.

(Lloyd look at this ex)

To install a callback routine, an application calls vbct_install_callbacks with type set to CS_SET and func as the address of the callback to install.

When a context is allocated, it has no callback routines installed. An application must specifically install any callbacks that are required.

When a connection is allocated, it picks up default callback routines from its parent context.

A connection picks up its parent context's callback routines only once, when it is allocated. This has two important implications:

- Existing connections are not affected by changes to their parent context's callback routines.

- If a callback routine of a particular type is de-installed for a connection, the connection does not pick up its parent context's callback routine. Instead, the connection is considered to have no callback routine of this type installed.

An application can use the CS_USERDATA property to transfer information between a callback routine and the program code that triggered it. The CS_USERDATA property allows an application to save user data in internal CT-Library space and retrieve it later.

See Also

vbct_deinstall_callbacks

vbct_labels

Define a security label or clear security labels for a connection.

Syntax

vbct_labels(connection&, action&, labelname$, labelvalue$, outlen&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

connection must represent a closed connection.

action&

One of the following symbolic values:

Value of action:	**vbct_labels:**
CS_SET	Sets a security label.
CS_CLEAR	Clears all security labels previously specified for this connection.

labelname$

If action is CS_SET, labelname points to the name of the security label being set.

If action is CS_CLEAR, labelname must be NULL.

Security label names must be at least one byte long and no more than CS_MAX_NAME bytes long.

If action is CS_CLEAR, pass namelen as CS_UNUSED.

labelvalue$

If action is CS_SET, labelvalue points to the value of the security label being set.

If action is CS_CLEAR, labelvalue must be NULL.

Security label values must be at least one byte long.

If action is CS_CLEAR, pass valuelen as CS_UNUSED.

outlen&

This parameter is currently unused and must be passed as NULL.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

Remarks

An application needs to define security labels if it will be connecting to a server that uses trusted-user security handshakes.

Secure SQL Server uses trusted-user security handshakes.   On Secure SQL Server, security labels are known as "sensitivity labels."

There are two ways for an application to define security labels.   An application can use either, or both, of these methods:

- The application can call vbct_labels one time for each label it wants to define.
- The application can call vbct_install_callbacks to install a user-supplied negotiation callback to generate security labels.   At connection time, CT-Library automatically triggers the callback in response to a request for security labels.

If an application uses both methods, the labels defined via vbct_labels and the labels generated by the negotiation callback are sent to the server at the same time.

A connection that will be participating in trusted-user security handshakes must set the CS_SEC_NEGOTIATE property to CS_TRUE.

There is no limit on the number of security labels that can be defined for a connection.

vbct_labels does not perform any type of checking on security labels, but simply passes the label name/label value combinations on to the server.

For example, vbct_labels does not raise an error if an application supplies two label values for the same label name.

See Also

vbct_install_callbacks, vbct_con_props_num, vbct_con_props_str, vbct_connect

vbct_options_num

Set, retrieve, or clear the values of server query-processing options.

Syntax

vbct_options_num(connection&, action&, option&, param$,outlen&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

connection is the server connection for which the option is set, retrieved, or cleared.

action&

One of the following symbolic values:

Value of action:	**vbct_options_num:**
CS_SET	Sets the option.
CS_GET	Retrieves the option. An application can use vbct_options_num to retrieve options from 10.0+ SQL Servers only.
CS_CLEAR	Clears the option by resetting it to its default value. Default values are determined by the server to which an application is connected.

option&

The server option of interest.

param&

All options take parameters.

When setting an option, param can point to a symbolic value, a boolean value, an integer value, or a character string.

When retrieving an option, param points to the space in which vbct_options_num places the value of the option.

If paramlen indicates that param is not large enough to hold the option's value, vbct_param sets outlen to the length of the value and returns CS_FAIL.

When clearing an option, param must be NULL.

outlen&

A pointer to an integer variable.

If an option is being set or cleared, outlen is not used and must be passed as NULL.

If an option is being retrieved, vbct_options_num sets outlen to the length, in bytes, of the option's value. This length includes a null terminator, if applicable.

If the option's value is larger than paramlen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Value of option:	param is:	Legal values for param:	Defaults to:
CS_OPT_ANSINULL	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_ANSIPERM	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_ARITHABORT	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_ARITHIGNORE	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_CHAINXACTS	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_CURCLOSEONXACT	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_DATEFIRST	A symbolic value representing the day to use as the first day of the week.	CS_OPT_SUNDAY CS_OPT_MONDAYCS_OPT_TUESDAY CS_OPT_WEDNESDAY, CS_OPT_THURSDAY, CS_OPT_FRIDAY, CS_OPT_SATURDAY	For us_english, the default is CS_OPT_SUNDAY.
CS_OPT_DATEFORMAT	A symbolic value representing the order of year, month, and day to be used in datetime values.	CS_OPT_FMTMDY,CS_OPT_FMTDMY,CS_OPT_FMTYMD,CS_OPT_FMTYDM,CS_OPT_FMTMYD,CS_OPT_ FMTDYM"	For us_english, the default is CS_OPT_FMTMDY.
CS_OPT_FIPSFLAG	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_FORCEPLAN	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_FORMATONLY	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_GETDATA	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_ISOLATION	A symbolic value representing the isolation level.	CS_OPT_LEVEL1, CS_OPT_LEVEL3	CS_OPT_LEVEL1
CS_OPT_NOCOUNT	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_NOEXEC	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_PARSEONLY	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_QUOTED_IDENT	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_RESTREES	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_ROWCOUNT	The maximum number of regular rows to return.	An integer value. 0 means all rows are returned.	0, meaning all rows are returned.
CS_OPT_SHOWPLAN	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_STATS_IO	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_STATS_TIME	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_STR_RTRUNC	A boolean value.	CS_TRUE, CS_FALSE	CS_FALSE
CS_OPT_TEXTSIZE	The length, in bytes, of the longest text or image value the server should return.	An integer value.	32,768 bytes.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. If vbct_options_num returns CS_FAIL, param remains untouched.

CS_CANCELED	The operation was canceled.
CS_PENDING	Asynchronous network I/O is in effect. Refer to Asychronous Programmingfor more information.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

Remarks

Although query-processing options can be set and cleared through the Transact-SQL set command, it is recommended that CT-Library applications use vbct_options_num instead. This is because vbct_options_num allows an application to check the status of an option, that cannot be done through the set command.

An application can use vbct_options_num to change server options only for a single connection at a time. The connection must be open and must have no active commands or pending results, but can have an open cursor.

See Also

vbct_capability, vbct_con_props_num, vbct_con_props_str

vbct_options_str

Set, retrieve, or clear the values of server query-processing options.

Syntax

vbct_options_str(connection&, action&, option&, param$,outlen&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

connection is the server connection for which the option is set, retrieved, or cleared.

action&

One of the following symbolic values:

Value of action:	**vbct_options:**
CS_SET	Sets the option.
CS_GET	Retrieves the option. An application can use vbct_options_str to retrieve options from 10.0+ SQL Servers only.
CS_CLEAR	Clears the option by resetting it to its default value. Default values are determined by the server to which an application is connected.

option&

The server option of interest.

param$

All options take parameters.

When setting an option, param can point to a symbolic value, a boolean value, an integer value, or a character string.

When retrieving an option, param points to the space in which vbct_options_str places the value of the option.

If paramlen indicates that param is not large enough to hold the option's value, vbct_param sets outlen to the length of the value and returns CS_FAIL.

When clearing an option, param must be NULL.

outlen&

A pointer to an integer variable.

If an option is being set or cleared, outlen is not used and must be passed as NULL.

If an option is being retrieved, vbct_options_str sets outlen to the length, in bytes, of the option's value. This length includes a null terminator, if applicable.

If the option's value is larger than paramlen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Value of option:	param is:	Legal values for param:	Defaults to:
CS_OPT_AUTHOFF	A string value representing an authority level.	A string value. Possible values include "sa", "sso", and "oper".	Not applicable
CS_OPT_AUTHON	A string value representing an authority level.	A string value. Possible values include "sa", "sso", and "oper".	Not applicable
CS_OPT_CURREAD	A string value representing a read level label.	A string value.	NULL
CS_OPT_CURWRITE	A string value representing a write level label.	A string value.	NULL
CS_OPT_IDENTITYOFF	A string value representing a table name.	A string value.	NULL
CS_OPT_IDENTITYON	A string value representing a table name.	A string value.	NULL

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. If vbct_options_str returns CS_FAIL, param remains untouched.
CS_CANCELED	The operation was canceled.
CS_PENDING	Asynchronous network I/O is in effect. Refer to Asychronous Programming for more information.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

Remarks

Although query-processing options can be set and cleared through the Transact-SQL set command, it is recommended that CT-Library applications use vbct_options_str instead. This is because vbct_options_str allows an application to check the status of an option, which cannot be done through the set command.

An application can use vbct_options_str to change server options only for a single connection at a time. The connection must be open and must have no active commands or pending results, but can have an open cursor.

See Also

vbct_capability, vbct_con_props_num, vbct_con_props_str

vbct_param

Define a command parameter.

Syntax

vbct_param(command&, datafmt(CS_DATAFMT), data(Any), datalen& indicator%)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

datafmt(CS_DATAFMT)

A pointer to a CS_DATAFMT structure that describes the parameter.

data(Any)

The address of the parameter data.

There are two ways to indicate a parameter with a null value:

- Pass indicator as -1. Data and datalen are ignored in this case.
- Pass data as NULL and datalen as 0 or CS_UNUSED.

datalen&

The length, in bytes, of the parameter data.

If datafmt.datatype indicates that the parameter is a fixed-length type, datalen is ignored. CS_VARBINARY and CS_VARCHAR are considered to be fixed-length types.

indicator%

An integer variable used to indicate a parameter with a null value. To indicate a parameter with a null value, pass indicator as -1. If indicator is -1, data and datalen are ignored.

Type of command:	**vbct_param called for what purpose?**	**datafmt.status is:**	**data, datalen are:**
Dynamic SQL execute	To pass input parameter values.	CS_INPUTVALUE	The parameter value and length.
Language	To pass input parameter values.	CS_INPUTVALUE	The parameter value and length.
Message	To pass input parameter values.	CS_INPUTVALUE	The parameter value and length.
RPC	To pass input or return parameter values.	CS_RETURN to pass a return parameter; CS_INPUTVALUE to pass a non-return parameter.	The parameter value and length.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.

CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

An application may need to call vbct_param:

- To identify update columns for a cursor declare command.
- To define host variable formats for a cursor declare command.
- To pass input parameter values for a cursor open, cursor update, dynamic SQL execute, language, message, or RPC command.

An application calls vbct_command to initiate a language, RPC or message command, and calls vbct_dynamic to initiate a Dynamic SQL execute command

In some cases an application may need to pass a parameter that has a value of NULL. For example, an application might pass parameters with null values to a stored procedure that assigns default values to NULL input parameters.

There are two ways to indicate a parameter with a null value:

- Pass indicator as -1. Data and datalen are ignored in this case.
- Pass data as NULL and datalen as 0 or CS_UNUSED.

CT-Library does not perform any conversion on parameters before passing them to the server. If parameter conversion is required, it is the server's responsibility.

Passing Input Parameter Values

An application may need to pass input parameter values for:

- CT-Library cursor open commands
- Dynamic SQL execute commands
- Language commands
- Message commands
- RPC commands

When passing input parameter values, parameters can either be named or unnamed. If one parameter is named, all parameters must be named. If parameters are not named, they are interpreted positionally.

Dynamic SQL execute commands require input parameter values when the prepared statement being executed contains dynamic parameter markers.

Language commands require input parameter values when the text of the language command contains host variables.

Message commands require input parameters values when the message takes parameters.

RPC commands require input parameter values when the stored procedure being executed takes parameters.

The following table lists the fields in datafmt that are used when passing input parameter values:

Field name:	**Set the field to:**

name	The name of the host variable.
namelen	The length, in bytes, of name, or 0 to indicate an unnamed parameter.
datatype	The datatype of the host variable. All standard CT-Library types are valid except for CS_TEXT_TYPE, CS_IMAGE_TYPE, and CT-Library user-defined types. If datatype is CS_VARCHAR_TYPE or CS_VARBINARY_TYPE then data must point to a CS_VARCHAR or CS_VARBINARY structure.
status	CS_INPUTVALUE
All other fields	Are ignored.

See Also

vbct_command_num, vbct_command_str, vbct_dynamic, vbct_send

vbct_poll

Poll connections for asynchronous operation completions and registered procedure notifications.

Syntax

vbct_poll(context&, connection&, milliseconds&, compid&, compstatus&)

Parameters

context&

The parameter passed is the context pointer that is obtained from the vbcs_ctx_alloc function.

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

miliseconds&

The length of time, in miliseconds, to wait for pending operations to complete.

compid&

The address of an integer variable. vbct_poll sets compid to one of the following symbolic vlaues to indicate what has completed:

Value of compid:	**Indicating:**
BLK_ROWXFER	vbblk_rowxfer has completed.
BLK_SENDROW	blk_sendrow has completed.
BLK_SENDTEXT	blk_sendtext has completed.
BLK_TEXTXFER	vbblk_textxfer has completed
CT_CANCEL	vbct_cancel has completed.
CT_CLOSE	vbct_close has completed.
CT_CONNECT	vbct_connect has completed.
CT_FETCH	vbct_fetch has completed.
CT_GET_DATA	vbct_get_data has completed.
CT_NOTIFICATION	A notification has been received.
CT_OPTIONS_NUM	vbct_options_num has completed.
CT_OPTIONS_STR	vbct_options_str has completed.
CT_RECVPASSTHRU	vbct_recvpassthru has completed.
CT_RESULTS	vbct_results has completed.
CT_SEND	vbct_send has completed.
CT_SEND_DATA	vbct_send_data has completed.
CT_SENDPASSTHRU	vbct_sendpassthru has completed.

A user-defined value. This value must be greater than or equal to CT_USER_FUNC. A user-defined function has completed.

comstatus&

A pointer to a variable type of CS_RETCODE. vbct_poll sets compstatus to indicate the final return code of the completed operation. This can be any of the return codes listed for the routine, with the exception of CS_PENDING.

context:	**connection:**	**compconn:**	**vbct_poll:**
NULL	Must have a value.	Must be NULL.	Checks the single connection specified by connection.
Has a value	Must be NULL.	Must have a value.	Checks all connections within this context. Sets compconn to point to the connection owning the first completed operation it finds.

tablename$

A pointer to the name of the table of interest.   Any legal server table name is acceptable.

Results

Returns:	**To Indicate:**
CS_SUCCEED	An operation has completed.
CS_FAIL	An error occurred.
CS_TIMED_OUT	The timeout value specified by milliseconds elapsed before any operation completed. Asynchronous operations may be in progress.
CS_QUIET	vbct_poll was called with milliseconds as 0 (to indicate that it should return immediately). No asynchronous operations are in progress, and no completed operations were found.
CS_INTERRUPT	A system interrupt has occurred.

Remarks

See Also

vbct_wakeup

vbct_recvpassthru

Receive a TDS (Tabular Data Stream) packet from a server.

Syntax

vbct_recvpassthru (command&, recvptr&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

recvptr&

The address of a pointer variable. vbct_recvpassthru sets the variable to the address of a buffer containing the most-recently-received TDS packet. The application is not responsible for allocating this buffer.

Results

Returns:	**To Indicate:**
CS_PASSTHRU_MORE	Packet received successfully; more packets are available.
CS_PASSTHRU_EOM	Packet received successfully; no more packets are available.
CS_FAIL	The routine failed.
CS_CANCELED	The pass-through operation was canceled.
CS_PENDI NG	Asynchronous network I/O is in effect. Refer to Asychronous Programming.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

Remarks

TDS is a communications protocol used for the transfer of requests and request results between clients and servers. Under ordinary circumstances, non-gateway applications do not usually have to deal with TDS, because CT-Library manages the data stream.

vbct_recvpassthru and vbct_sendpassthru are useful in gateway applications. When an application serves as the intermediary between two parties (such as a client and a remote server, or two servers), it can use these routines to pass the TDS stream from one server to the other, eliminating the process of interpreting the information and re-encoding it.

vbct_recvpassthru reads a packet of bytes from a server connection and sets recvptr to point to the buffer containing the bytes.

Default packet sizes vary by platform. On most platforms, a packet has a default size of 512 bytes. A connection can change its packet size via vbct_con_props_num and vbct_con_props_str.

vbct_recvpassthru returns CS_PASSTHRU_EOM if the TDS packet has been marked by the server as

EOM (End Of Message).   If the TDS packet is not marked EOM, vbct_recvpassthru returns CS_PASSTHRU_MORE.

A connection which is being used for a pass-through operation cannot be used for any other CT-Library function until CS_PASSTHRU_EOM has been received.

See Also

vbct_getloginfo, vbct_sendpassthru, vbct_setloginfo

vbct_remote_pwd

Define or clear passwords to be used for server-to-server connections.

Syntax

vbct_remote_pwd(connection&, action&, server_name$, password$)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

It is illegal to define remote passwords for a connection that is open.

action&

One of the following symbolic values:

Value of action:	**vbct_remote_pwd:**
CS_SET	Sets the remote password
CS_CLEAR	Clears all remote passwords specified for this connection by setting them to NULL

server_name$

A pointer to the name of the server for which the password is being defined. server_name is the name given to the server in an interfaces file.

If server_name is NULL, the specified password will be considered a "universal" password that can be used with any server that does not have a password explicitly specified for it.

If action is CS_CLEAR, server_name must be NULL.

password$

A pointer to the password being installed for remote logins to the server_name server.

If action is CS_CLEAR, password must be NULL.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

vbct_remote_pwd defines the password that a server will use when logging into another server.

A Transact-SQL language command or stored procedure running on one server can call a stored procedure located on another server. To accomplish this server-to-server communication, the first server to which an application has connected via vbct_connect , actually logs into the second, remote server, performing a server-to-server remote procedure call.

vbct_remote_pwd allows an application to specify the password to be used when the first server logs into the remote server.

Multiple passwords can be specified; one for each server that a server might need to log into. Each password must be defined with a separate call to vbct_remote_pwd.

If an application does not specify a remote password for a particular server, the password defaults to the password set for this connection via vbct_con_props_num and vbct_con_props_str, if any. If a password has not been defined, the password defaults to NULL. If an application user generally has the same password on different servers, this default behavior may be acceptable.

Remote passwords are stored in an internal buffer which is only 255 bytes long. Each password's entry in the buffer consists of the password itself, the associated server name, and two extra bytes. If the addition of a password to this buffer causes overflow, vbct_remote_pwd returns CS_FAIL and generates a CT-Library error message that indicates the problem.

It is an error to call vbct_remote_pwd to define a remote password for a connection that is already open. Define remote passwords before calling vbct_connect to create an active connection.

An application can call vbct_remote_pwd to clear remote passwords for a connection at any time.

See Also

vbct_con_props_num, vbct_con_props_str, vbct_connect

vbct_res_info

Retrieve current result set or command information.

Syntax

vbct_res_info(command&, type&, info&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

type&

The type of information to return.

info&

A pointer to the space in which vbct_res_info will place the requested information.

ummary of Parameters Table

Value of type:	**vbct_res_info returns:**	**The information is available after vbct_results sets its result_type parameter to:**	**buffer is set to:**
CS_BROWSE_INFO	CS_TRUE if browse-mode information is available; CS_FALSE if browse-mode information is not available.	CS_ROW_RESULT	CS_TRUE or CS_FALSE.
CS_CMD_NUMBER	The number of the command that generated the current result set.	Any value.	An integer value.
CS_MSGTYPE	An integer representing the id of the message that makes up the current result set.	CS_MSG_RESULT	A small integer.
CS_NUM_COMPUTES	The number of compute clauses in the current command.	CS_COMPUTE_RESULT	An integer value.
CS_NUMDATA	The number of items in the current result set.	CS_COMPUTE_RESULT, CS_COMPUTEFMT_RESULT, CS_CURSOR_RESULT, CS_DESCRIBE_RESULT, CS_PARAM_RESULT, CS_ROW_RESULT, CS_ROWFMT_RESULT, CS_STATUS_RESULT	An integer value.
CS_NUMORDER COLS	The number of columns specified in the order-by clause of the current	CS_ROW_RESULT	An integer value.

	command.		
CS_ORDERBY_COLS	The select list id numbers of columns specified in a the order by clause of the current command.	CS_ROW_RESULT	An array of integers.
CS_ROW_COUNT	The number of rows affected by the current command.	CS_CMD_DONE, CS_CMD_FAIL CS_CMD_SUCCEED	An integer value.
CS_TRANS_STATE	The current server transaction state.	Any value.	A symbolic value.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. vbct_res_info returns CS_FAIL if the requested information is larger than buflen bytes, or if there is no current result set.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer Asychronous Programming.

Remarks

vbct_res_info returns information about the current result set or the current command. The current command is defined as the command that generated the current result set.

A result set is a collection of a single type of result data. Result sets are generated by commands.

Most typically, an application calls vbct_res_info with type as CS_NUMDATA, to determine the number of items in a result set.

Determining Whether Browse Mode Information is Available

To determine whether browse-mode information is available, call vbct_res_info with type as CS_BROWSE_INFO.

If browse-mode information is available, an application can call ct_br_column and ct_br_table to retrieve the information. If browse-mode information is not available, calling ct_br_column or ct_br_table will result in a CT-Library error.

Retrieving the Command Number for Current Results

To determine the number of the command that generated the current results, call vbct_res_info with type as CS_CMD_NUMBER.

CT-Library keeps track of the command number by counting the number of times vbct_results returns CS_CMD_DONE.

An application's first call to vbct_results following a vbct_send call sets the command number to 1. After this, it is incremented each time vbct_results is called after returning CS_CMD_DONE.

CS_CMD_NUMBER is useful in the following cases:

- To find out which Transact-SQL command within a language command generated the current result set.

- To find out which cursor command, in a batch of cursor commands, generated the current result set.
- To find out which select command in a stored procedure generated the current result set.

A language command contains a string of Transact-SQL text. This text represents one or more Transact-SQL commands. When used against a language command, "command number" refers to the number of the Transact-SQL command in the language command.

For example, the string:

```
select * from authors
select * from titles
insert newauthors
select *
from authors
where city = "San Francisco"
```

represents three Transact-SQL commands, two of which can generate result sets. In this case, the command number that vbct_res_info returns can be from 1 to 3, depending on when vbct_res_info is called.

Inside stored procedures, only select statements cause the command number to be incremented. If a stored procedure contains seven Transact-SQL commands, three of which are selects, the command number that vbct_res_info returns can be any integer from 1 to 3, depending on which select generated the current result set.

The command number that vbct_res_info returns can be 1, 2, or 3, depending on which cursor command generated the current result type.

Retrieving a Message ID

To retrieve a message id, call vbct_res_info with type as CS_MSGTYPE.

Servers can send messages to client applications. Messages are received in the form of "message result sets." Message result sets contain no fetchable data, but rather have an id number.

Messages can also have parameters. Message parameters are returned to an application as a parameter result set, immediately following the message result set.

Retrieving the Number of Compute Clauses

To determine the number of compute clauses in the command that generated the current result set, call vbct_res_info with type as CS_NUM_COMPUTES.

A Transact-SQL select statement can contain compute clauses that generate compute result sets.

Retrieving the Number of Result Data Items

To determine the number of result data items in the current result set, call vbct_res_info with type as CS_NUMDATA..

Results sets contain result data items. Row, cursor, and compute result sets contain columns, a parameter result set contains parameters, and a status result set contains a status. The columns, parameters, and status are known as "result data items".

A message result set does not contain any data items.

Retrieving the Number of Columns in an Order-By Clause

To determine the number of columns in a Transact-SQL select statement's order by clause, call vbct_res_info with type as CS_NUMORDERCOLS.

A Transact-SQL select statement can contain an order by clause, which determines how the rows resulting from the select are ordered on presentation.

Retrieving the Column ID's of Order-By Columns

To get the select list column id's of order-by columns, call vbct_res_info with type as CS_ORDERBY_COLS.

Columns named in an order by clause must also be named in the select list of the select statement. Columns in a select list have a "select list id," which is the number in which they appear in the list. For example, in the following query, au_lname and au_fname have select list id's of 1 and 2 respectively:

```
    select au_lname, au_fname from authors
        order by au_fname, au_lname
```

Given the preceding query, the call:

```
    vbct_res_info(cmd, CS_ORDERBY_COLS, myspace, 8,
  outlength)
```

sets myspace to an array of two CS_INTs containing the integers 2 and 1.

Retrieving the Number of Rows for the Current Command

To determine the number of rows affected by the current command, call vbct_res_info with type as CS_ROW_COUNT.

An application can retrieve a row count after vbct_results sets its result_type parameter to CS_CMD_SUCCEED, CS_CMD_DONE, or CS_CMD_FAIL. A row count is guaranteed to be accurate if vbct_results has just set result_type to CS_CMD_DONE.

If the command is one that executes a stored procedure, for example a Transact-SQL exec language command or a remote procedure call command, vbct_res_info sets buffer to the number of rows affected by the last statement in the stored procedure that affects rows.

vbct_res_info sets buffer to CS_NO_COUNT if any of the following are true:

- The Transact-SQL command fails for any reason, such as a syntax error.
- The command is one that never affects rows, such as a Transact-SQL print command.
- The command executes a stored procedure that does not affect any rows.
- The CS_OPT_NOCOUNT option is on.

Retrieving the Current Server Transaction State

To determine the current server transaction state, call vbct_res_info with type as CS_TRANS_STATE.

See Also

vbct_cmd_props_num, vbct_cmd_props_str, vbct_con_props_num, vbct_con_props_str, vbct_results

vbct_results

Set up result data to be processed.

Syntax

vbct_results(command&, result_type&)

Parameters

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

result_type&

A pointer to an integer variable which vbct_results sets to indicate the current type of result.

The following table lists the possible values of result_type:

Value of result_type:	What it indicates:	Result set contains:
CS_CMD_DONE	The results of a logical command have been completely processed.	Not applicable.
CS_CMD_FAIL	The server encountered an error while executing a command.	No results.
CS_CMD_SUCCEED	The success of a command that returns no data, such as a language command containing a Transact-SQL insert statement.	No results.
CS_COMPUTE_RESULT	Compute row results.	A single row of compute results.
CS_CURSOR_RESULT	Cursor row results.	Zero or more rows of tabular data.
CS_PARAM_RESULT	Return parameter results.	A single row of return parameters.
CS_ROW_RESULT	Regular row results.	Zero or more rows of tabular data.
CS_STATUS_RESULT	Stored procedure return status results.	A single row containing a single status.
CS_COMPUTEFMT_RESULT	Compute format information.	No fetchable results. An application can call vbct_describe, vbct_res_info, and vbct_compute_info to retrieve compute format information.
CS_ROWFMT_RESULT	Row format information.	No fetchable results. An application can call vbct_describe and vbct_res_info to retrieve row format information.
CS_MSG_RESULT	Message arrival.	No fetchable results. An application can call vbct_res_info to get the message's id. Parameters

		associated with the message, if any, are returned as a separate parameter result set.
CS_DESCRIBE_RESULT	Dynamic SQL descriptive information.	No fetchable results. An application can call vbct_describe or vbct_dyndesc to retrieve the information

Results

Returns:	**To Indicate:**
CS_SUCCEED	A result set is available for processing.
CS_END_RESULTS	All results have been completely processed.
CS_FAIL	The routine failed; any remaining results are no longer available. If vbct_results returns CS_FAIL, an application must call vbct_cancel with type as CS_CANCEL_ALL before using the affected command structure to send another command. If vbct_cancel returns CS_FAIL, the application must call vbcs_close(CS_FORCE_CLOSE) to force the connection closed.
CS_CANCELED	Results have been canceled.
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

Remarks

An application calls vbct_results after sending a command to the server via vbct_send, and before reading the results of that command (if any) via vbct_fetch.

"Result data" is an umbrella term for all the types of data that a server can return to an application. These types of data include:

- Regular rows
- Cursor rows
- Return parameters
- Stored procedure return status numbers
- Compute rows
- Dynamic SQL descriptive information
- Regular row and compute row format information
- Messages

vbct_results is used to set up all of these types of results for processing.

NOTE: Don't confuse message results with server error and informational messages. Refer to Error and Message Handling for a discussion of error and informational messages.

Result data is returned to an application in the form of a "result set". A result set includes only a single type of result data. For example, a regular row result set contains only regular rows, and a return parameter result set contains only return parameters.

The vbct_results Loop

Because a command can generate a result set that spans multiple buffers, an application must call vbct_results as long as it continues to return CS_SUCCEED, indicating that results are available. The simplest way to do this is in a loop that terminates when vbct_results fails to return CS_SUCCEED. After the loop, an application can use a case-type statement to test vbct_results' final return code to determine why the loop terminated.

Results are returned to an application in the order in which they are produced. However, this order is not always easy to predict. For example, when an application calls a stored procedure that in turn calls another stored procedure, the application might receive a number of regular row and compute row result sets, as well as a return parameter and a return status result set. The order in which these results are returned depends on how the stored procedures are written.

For this reason, it is recommended that an application's vbct_results loop be coded so that control drops into a case-type statement that handles all types of results that can be received. The return parameter result_type indicates symbolically what type of result data the result set contains.

When are the Results of a Command Completely Processed?

vbct_results sets result_type to CS_CMD_DONE to indicate that the results of a "logical command" have been completely processed.

A logical command is defined as any command defined via vbct_command, or vbct_dynamic with the following exceptions:

- Each Transact-SQL select statement inside a stored procedure is a logical command. Other Transact-SQL statements inside stored procedures do not count as logical commands.
- Each Transact-SQL statement executed by a dynamic SQL command is a distinct logical command.
- Each Transact-SQL statement in a language command is a logical command.

A result_type of CS_CMD_SUCCEED or CS_CMD_FAIL is immediately followed by a result_type of CS_CMD_DONE.

A connection has pending results if it has not processed all of the results generated by a CT-Library command. Usually, an application cannot send a new command on a connection with pending results.

Canceling Results

To cancel all remaining results from a command (and eliminate the need to continue calling vbct_results until it fails to return CS_SUCCEED), call vbct_cancel with type as CS_CANCEL_ALL.

To cancel only the current results, call vbct_cancel with type as CS_CANCEL_CURRENT.

Special Kinds of Result Sets

A message result set contains no actual result data, rather, a message has an "id". An application can call vbct_res_info to get a message's id. In addition to an id, messages can have parameters. Message parameters are returned to an application as a parameter result set, immediately following the message result set.

Row format and compute format result sets contains no actual result data. Instead, format result sets contain formatting information for the regular row or compute row result sets with which they are associated.

This type of format information is of use primarily in gateway applications, which need to repackage SQL Server format information before sending it to a foreign client. After vbct_results indicates format results, a gateway application can retrieve format information by calling vbct_describe, vbct_res_info, and vbct_compute_info.

All format information for a command is returned before any data. For example, the row format and compute format result sets for a command precede the regular row and compute row result sets generated by the command.

An application will not receive format results unless the CT-Library CS_EXPOSE_FMTS property is set to CS_TRUE.

A describe result set contains no actual result data. Instead, a describe result set contains descriptive information generated by a dynamic SQL describe input or describe output command. After vbct_results indicates describe results, an application can retrieve information by calling vbct_describe and vbct_dyndesc.

vbct_results and Stored Procedures

A run-time error on a language command containing an execute statement will generate a result_type of CS_CMD_FAIL. For example, this occurs if the procedure named in the execute statement cannot be found.

A run-time error on a statement inside a stored procedure will not generate a CS_CMD_FAIL. For example, if the stored procedure contains an insert statement and the user does not have insert permission on the database table, the insert statement will fail, but vbct_results will still return CS_SUCCEED. To check for run-time errors inside stored procedures, examine the procedure's return status number, which is returned as a return status result set immediately following the row and parameter results, if any, from the stored procedure. If the error generates a server message, it is also available to the application.

See Also

vbct_command_num, vbct_command_str, vbct_describe, vbct_dynamic, vbct_fetch, vbct_send

vbct_send

Send a command to the server.

Syntax

vbct_send(command&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the the vbct_con_alloc function.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. For less serious failures, the application can call vbct_cancel(CS_CANCEL_ALL) to clean up the command structure. For more serious failures, the application must call vbcs_close (CS_FORCE_CLOSE) to force the connection closed.
CS_CANCELED	The routine was canceled.
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

Sending a command to a server is a four step process. To send a command to a server, an application must:

- Initiate the command by calling vbct_command_num, vbct_command_str, or vbct_dynamic. These routines set up internal structures that are used in building a command stream to send to the server.
- Pass parameters for the command (if required) by calling vbct_param once for each parameter that the command requires.

Not all commands require parameters. For example, a remote procedure call command may or may not require parameters, depending on the stored procedure being called.

- Send the command to the server by calling vbct_send.
- Verify the success of the command by calling vbct_results.

This last step does not imply that an application need only call vbct_results once. An application needs to continue calling vbct_results until it no longer returns CS_SUCCEED.

An application can call vbct_cancel(CS_CANCEL_ALL) to cancel a command that has been initiated but not yet sent.

Once an application has sent a command, it must call vbct_results before calling vbct_cancel to cancel the command.

vbct_send uses an asynchronous write and not does wait for a response from the server.   An application must call vbct_results to verify the success of the command and to set up the command results for processing

See Also

vbct_command_num, vbct_command_str, vbct_dynamic, vbct_fetch, vbct_param , vbct_results

vbct_send_data

Send a chunk of text or image data to the server.

Syntax

vbct_send_data(command& buffer(Any), buflen&)

Parameters

command&

The parameter passed is the command pointer which is passed from the vbct_cmd_alloc.

A pointer to the value to write to the server.

buffer(Any)

A pointer to the value to write to the server.

buflen&

The length, in bytes, of buffer.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_CANCELED	The send data operation was canceled.
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.
CS_BUSY	An asynchronous operation is already pending for this connection. For more information, refer to Asychronous Programming.

Remarks

An application can use vbct_send_data to write a text or image value to a database column. This writing operation is actually an update; the column must have a value when vbct_send_data is called to write a new value.

This is because vbct_send_data uses text timestamp information when writing to the column, and a column does not have a valid text timestamp until it contains a value. The value contained in the text or image column can be NULL, but the NULL must be entered explicitly with the SQL update statement.

In order to perform a send-data operation, an application must have a current I/O descriptor, or CS_IODESC structure, describing the column value that will be updated:

- The textptr field of the CS_IODESC identifies the target column.

- The timestamp field of the CS_IODESC is the text timestamp of the column value. If timestamp does not match the current database text timestamp for the value, the update operation will fail.

- The total_txtlen field of the CS_IODESC indicates the total length, in bytes, of the column's new value.

An application must call vbct_send_data in a loop to write exactly this number of bytes before calling vbct_send to indicate the end of the text or image update operation.

- The log_on_update of the CS_IODESC tells the server whether or not to log the update operation.

- The locale field of the CS_IODESC points to a CS_LOCALE structure that contains localization information for the new value, if any.

A typical application will change only the values of the locale, total_txtlen, and log_on_update fields before using an I/O descriptor in an update operation, but an application that is updating the same column value multiple times will need to change the value of the timestamp field as well.

A successful text or image update generates a parameter result set that contains the new text timestamp for the text or image value.   If an application plans to update the text or image value a second time, it must save this new text timestamp and copy it into the CS_IODESC for the value before calling vbct_data_info to define the CS_IODESC for the update operation.

A text or image update operation is equivalent to a language command containing a Transact-SQL update statement.

The command space identified by cmd must be idle before a text or image update operation is initiated. A command space is idle if there are no active commands, pending results, or open cursors in the space.

See Also

vbct_data_info, vbct_get_data

vbct_sendpassthru

Send a TDS (Tabular Data Stream) packet to a server.

Syntax

vbct_sendpassthru (command&, sendptr&)

Parameters

command&

The parameter passed is the command pointer which is passed from the vbct_cmd_alloc function.

sendptr&

A pointer to a buffer containing the TDS packet to be sent to the server.

Results

Returns:	**To Indicate:**
CS_PASSTHRU_MORE	Packet sent successfully; more packets are available.
CS_PASSTHRU_EOM	Packet sent successfully; no more packets are available.
CS_FAIL	The routine failed.
CS_CANCELLED	The routine was cancelled.
CS_PENDING	Asynchronous network I/O is in effect. Refer to Asychronous Programming.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

TDS is a communications protocol used for the transfer of requests and request results between clients and servers. Under ordinary circumstances, non-gateway applications do not usually have to deal with TDS, because CT-Library manages the data stream.

vbct_recvpassthru and vbct_sendpassthru are useful in gateway applications. When an application serves as the intermediary between two parties (such as a client and a remote server, or two servers), it can use these routines to pass the TDS stream from one server to the other, eliminating the process of interpreting the information and re-encoding it.

vbct_sendpassthru sends a packet of bytes from the sendptr buffer. Most commonly, sendptr will be recvptr as returned by srv_recvpassthru. sendptr can also be the address of a user-allocated buffer containing the packet to send.

Default packet sizes vary by platform. On most platforms, a packet has a default size of 512 bytes. A connection can change its packet size via vbct_con_props_num and vbct_con_props_str.

vbct_sendpassthru returns CS_PASSTHRU_EOM if the TDS packet in the buffer is marked EOM (End Of Message). If the TDS packet is not marked EOM, vbct_sendpassthru returns CS_PASSTHRU_MORE.

A connection which is being used for a pass-through operation cannot be used for any other CT-Library function until CS_PASSTHRU_EOM has been received.

See Also

vbct_getloginfo, vbct_recvpassthru, vbct_setloginfo

vbct_setloginfo

Transfer TDS login response information from a CS_LOGINFO structure to a CS_CONNECTION structure.

Syntax

vbct_setloginfo (connection&, loginfo&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

loginfo&

A pointer to a CS_LOGINFO structure.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

TDS (Tabular Data Stream) is a communications protocol used for the transfer of requests and request results between clients and servers.

Because vbct_setloginfo frees the CS_LOGINFO structure after transferring the TDS information, an application cannot re-use the CS_LOGINFO. An application can get a new CS_LOGINFO by calling vbct_getloginfo.

There are two reasons an application might call vbct_setloginfo:

- If it is an Open Server gateway application using TDS pass-through.
- In order to copy login properties from an open connection to a newly-allocated connection structure.

NOTE: Do not call vbct_setloginfo from within a completion callback routine. vbct_setloginfo calls system-level memory functions which may not be re-entrant.

TDS Pass-Through

When a client connects directly to a server, the two programs negotiate the TDS format they will use to send and receive data. When a gateway application uses TDS pass-through, the gateway forwards TDS packets between the client and a remote server without examining or processing them. For this reason, the remote server and the client must agree on a TDS format to use.

See Also

vbct_getloginfo, vbct_recvpassthru, vbct_sendpassthru

vbct_wakeup

Call a connection's completion callback.

Syntax

vbct_wakeup(connection&, cmd&, function&, status&)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

Either connection or cmd must be non-NULL.

If connection is supplied, it is passed as the connection parameter to the completion callback. If connection is NULL, cmd's parent connection is passed as the connection parameter to the completion callback.

command&

The parameter passed is the command pointer that is obtained from the vbct_cmd_alloc function.

Either connection or cmd must be non-NULL.

If connection is NULL, cmd's parent connection's completion callback is called.

cmd is passed as the command parameter to the completion callback. If cmd is NULL then NULL is passed for the command parameter.

function&

A symbolic value indicating which routine has completed. function can be a user-defined value. function is passed as the function parameter to the completion callback. The following table lists the symbolic values that are legal for function:

Value of function:	**To indicate:**
BLK_ROWXFER	vbblk_rowxfer has completed.
BLK_SENDROW	vbblk_sendrow has completed.
BLK_SENDTEXT	vbblk_sendtext has completed.
BLK_TEXTXFER	vbblk_textxfer has completed
CT_CANCEL	vbct_cancel has completed.
CT_CLOSE	vbct_close has completed.
CT_CONNECT	vbct_connect has completed.
CT_FETCH	vbct_fetch has completed.
CT_GET_DATA	vbct_get_data has completed.
CT_OPTIONS	vbct_options has completed.
CT_RECVPASSTHRU	vbct_recvpassthru has completed.
CT_RESULTS	vbct_results has completed.
CT_SEND	vbct_send has completed.
CT_SEND_DATA	vbct_send_data has completed.
CT_SENDPASSTHRU	vbct_sendpassthru has completed.

A user-defined value. This value must be greater than or equal to CT_USER_FUNC.	A user-defined function has completed.

status&

The return status of the completed routine. This value is passed as the status parameter to the completion callback.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_BUSY	An asynchronous operation is already pending for this connection. Refer to Asychronous Programming.

Remarks

vbct_wakeup is intended for use in applications that create an asynchronous layer on top of CT-Library.

An application cannot call vbct_wakeup if the CS_DISABLE_POLL property is set to CS_TRUE.

Refer to Callbacks.

Refer to Asychronous Programming for more information on using vbct_wakeup in asynchronous CT-Library applications.

See Also

vbct_install_callbacks

vbblk_alloc

Allocate a CS_BLKDESC structure.

Syntax

vbblk_alloc(connection& version&, blk_desc&r)

Parameters

connection&

The parameter passed is the connection pointer that is obtained from the vbct_con_alloc function.

This connection must not have any pending results.

version&

The version of bulk copy behavior that the application expects.   During bulk copy initialization, version's value is checked for compatibility with CT-Library's version level.

Currently, BLK_VERSION_100 is the only value that is legal for version.

blkdesc&

The parameter is primed with the bulk copy description pointer.   This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

In case of error, vbblk_alloc sets blk_pointer to NULL.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a vbblk_alloc failure is a lack of adequate memory.

Remarks

A CS_BLKDESC structure, also called a "bulk descriptor structure" contains information about a particular bulk copy operation.

Before calling vbblk_alloc, an application must call the CT-Library routines vbct_con_alloc and vbct_connect to allocate a CS_CONNECTION structure and open the connection.

vbblk_alloc must be the first routine called in a bulk copy operation.

To de-allocate a CS_BLKDESC structure, an application can call vbblk_drop.

See Also

vbblk_init, vbct_con_alloc, vbct_connect

vbblk_bind

Bind a program variable and a database column.

Syntax

vbblk_bind(blkdesc&, column&, datafmt(DATAFMT))

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer. This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

column&

The number of the column to bind to the program variable.

The first column in a table is column number 1, the second number 2, and so forth.

datafmt(DATAFMT)

A pointer to the CS_DATAFMT structure that describes the program variable to bind to the column.

The chart below lists the fields in datafmt that are used by blk_bind, and contains general information about the fields. vbblk_bind ignores fields that it does not use:

Field name:	When is the field used?	Set the field to:
name	Not used.	Not applicable.
namelen	Not used.	Not applicable.
datatype	Always.	A type constant (CS_xxx_TYPE) representing the datatype of the program variable. All type constants listed on the Types topics page in the Open Client CT-Library Reference Manual are valid. Open Client user-defined types are not valid. vbblk_bind supports a wide range of type conversions, so datatype can be different from the column's type. For instance, by specifying a variable type of CS_FLOAT_TYPE, a money column can be bound to a CS_FLOAT program variable. The appropriate data conversion happens automatically. For a list of the data conversions provided by CT-Library, see the manual page for vbcs_will_convert. If datatype is CS_BOUNDARY_TYPE or CS_SENSITIVITY_TYPE, the buffer program variable must be of type CS_CHAR.
format	When binding to character- or binary-type destination variables on copies out from the database; otherwise CS_FMT_UNUSED.	A bit-mask of the following symbols: For character and text destinations only: CS_FMT_NULLTERM to null-terminate the data, or CS_FMT_PADBLANK to pad to the full length of the variable with spaces. For character, binary, text, and image destinations: CS_FMT_PADNULL to pad to

		the full length of the variable with nulls. For any type of destination: CS_FMT_UNUSED if no format information is being provided.
maxlength	When binding non- fixed-length variables for copies out from the database. When binding fixed-length variables, maxlength is ignored.	The maximum length of the buffer program variable. When binding character or binary variables, maxlength must describe the total maximum length of the program variable, including any space required for special terminating bytes, such as a null terminator. For a bulk copy into the database, maxlength is the maximum length of the data that will be copied from the buffer program variable. For a bulk copy out from the database, maxlength is the length of the buffer program variable.
scale	Only when binding to numeric or decimal variables.	The scale of the program variable. If the source data is the same type as the destination, then scale can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for scale from the source data. scale must be less than or equal to precision.
precision	Only when binding numeric or decimal destinations.	The precision of the program variable. If the source data is the same type as the destination, then precision can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for precision from the source data. precision must be greater than or equal to scale.
status	Not used.	Not applicable.
count	Always.	count is the number of rows to transfer per vbblk_rowxfer call. If count is greater than 1, array binding is considered to be in effect. On copies out from the database, if count is larger than the number of available rows, only the available rows are copied. count must have the same value for all columns being transferred, with one exception: an application can intermix counts of 0 and 1. This is because when count is 0, 1 row is transferred.
usertype	Not used.	Not applicable.
locale	If supplied, locale is used. Otherwise, default localization applies.	A pointer to a CS_LOCALE structure containing locale information for the buffer program variable.

Copies Into the Database

When calling vbblk_bind to:	**buffer is:**	**datalen is:**
Bind a normal value.	A pointer to a program variable.	The length of the data in buffer.
Indicate that a column value will be transferred using vbblk_textxfer.	NULL	The total length of the data that will be sent using vbblk_textxfer. In this case, datafmt.maxlength is ignored.
Instruct bulk copy to use the column's default value. If no default value is available for the column, NULL values are	Ignored	0

inserted.		
Set up array binding.	A pointer to an array of program variables.	A pointer to an array containing the lengths of the data in the buffer variables.

Copies Out From the Database

When calling vbblk_bind to:	buffer is:	datalen is:
Bind a normal value.	Non-null	The length of the buffer data space.
Indicate that a column value will be transferred using vbblk_textxfer.	NULL	Ignored. In this case, datafmt.maxlength represents the length of the buffer data space

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. vbblk_bind returns CS_FAIL if the application has not called vbblk_init to initialize the bulk copy operation.

Remarks

vbblk_bind is a CT-Library bulk copy routine.

When copying into a database, an application must call vbblk_bind once for each column in the database table. When copying out, an application does not need to call vbblk_bind for columns in which it has no interest.

To indicate that a column value will be transferred via vbblk_textfer, an application calls vbblk_bind with buffer as NULL. A typical application will only use vbblk_textxfer to transfer large text or image values.

An application can call vbblk_bind in between calls to vbblk_rowxfer to reflect changes, if any, in a variable's address or length. If an application calls vbblk_bind multiple times for a single column or variable, only the last binding takes effect.

An application can call vbblk_describe to get a CS_DATAFMT description of a particular column.

Bulk Copies Into the Database

When copying in, an application can instruct bulk copy to use a column's default value by setting datalen to 0. If no default value is defined for the column, NULL values are inserted.

Clearing Bindings

To clear a binding, call vbblk_bind with buffer, datafmt, datalen, and indicator as NULL. Otherwise, bindings remain in effect until an application calls vbblk_done with type as CS_BLK_ALL to indicate that the bulk copy operation is complete.

To clear all bindings, pass colnum as CS_UNUSED, with buffer, datafmt, datalen, and indicator as NULL. An application typically clears all bindings when it needs to change the count that is being used for array binding.

Array Binding

Array binding is the process of binding a column to an array of program variables.   At row transfer time, multiple rows' worth of the column are transferred either to or from the array of variables with a single vbblk_rowxfer call.   An application indicates array binding by setting datafm.count to a value greater than 1.

If array binding is in effect, then each of buffer, datalen, and indicator must point to arrays.   Each length and indicator variable describes the corresponding data in the buffer array.

When array binding, all vbblk_bind calls must use the same value for   datafmt. count.   For example, when binding three columns to arrays, it is an error to use a count of five in your first two vbblk_bind calls and a count of three in the last.

However, an application can intermix counts of 0 and 1. counts of 0 and 1 are considered to be equivalent because they both cause vbblk_rowxfer to transfer a single row.

If array binding is in effect, an application cannot use vbblk_textxfer to transfer data.

See Also

vbblk_describe, vbblk_init

vbblk_describe

Retrieve a description of a database column.

Syntax

vbblk_describe(blkdesc&, column&, datafmt(DATAFMT), prlength&)

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer. This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

column&

The number of the column of interest. The first column in a table is column number 1, the second is number 2, and so forth.

datafmt(DATAFMT)

A pointer to a CS_DATAFMT structure. vbblk_describe fills datafmt with a description of the database column referenced by colnum.

For a bulk copy into the database, vbblk_describe fills in the following fields in the CS_DATAFMT:

Field name:	**vbblk_describe sets the field to:**
name	The null-terminated name of the column, if any. A NULL name is indicated by a namelen of 0.
namelen	The actual length of the name, not including the null terminator. 0 to indicate a NULL name.
datatype	A type constant representing the datatype of the column. All type constants listed on the Types topics page are valid, with the exceptions of CS_VARCHAR_TYPE and CS_VARBINARY_TYPE.
maxlength	The maximum possible length of the data for the column.
scale	The scale of the column.
precision	The precision of the column.

For a bulk copy out from the database, vbblk_describe fills in the following fields in the CS_DATAFMT:

Field name:	**vbblk_describe sets the field to:**
name	The null-terminated name of the column, if any. A NULL name is indicated by a namelen of 0.
namelen	The actual length of the name, not including the null terminator. 0 to indicate a NULL name.
datatype	The datatype of the column. All datatypes listed on the types topics page are valid.
maxlength	The maximum possible length of the data for the column.
scale	The scale of the column.
precision	The precision of the column.
status	A bitmask of the following symbols, or-ed together: CS_CANBENULL to indicate that the column can contain NULL values. CS_HIDDEN to indicate that this column is a "hidden" column that has been exposed. CS_KEY to indicate the column is

	part of the key for a table. CS_VERSION_KEY to indicate the column is part of the version key for the row.
usertype	The SQL Server user-defined datatype of the column, if any. usertype is set in addition to (not instead of) datatype.
locale	A pointer to a CS_LOCALE structure that contains locale information for the data

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. vbblk_describe returns CS_FAIL if colnum does not represent a valid result column.

Remarks

vbblk_describe is a CT-Library bulk copy routine.

An application typically uses a column description in order to help determine compatible program variable types and sizes.

See Also

vbblk_init

vbblk_done

Mark a complete bulk copy operation or a complete bulk copy batch.

Syntax

vbblk_done(blkdesc&, type&, outrow&)

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer. This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

type&

The type of operation. The following table lists the symbolic values that are legal for type:

Value of type:	**vbblk_done:**
CS_BLK_ALL	Marks a complete bulk copy in or bulk copy out operation.
CS_BLK_BATCH	Marks a complete "batch" in a batched bulk copy in operation.
CS_BLK_CANCEL	Cancels a bulk copy batch or bulk copy operation.

outrow&

A pointer to an integer variable. vbblk_done sets outrow to the number of rows bulk copied to SQL Server since the application's last vbblk_done call.

Results

Returns	**To Indicate**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. Common reasons for failure include an invalid blkdesc pointer or an illegal value for type
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.

Remarks

vbblk_done is both a CT-Library and Server-Library bulk copy routine.

Calling vbblk_done with type as CS_BLK_ALL marks the end of a bulk copy operation. Once an application marks the end of a bulk copy operation, it cannot call any bulk copy routines (except for vbblk_drop and vbblk_alloc) until it initializes a new bulk copy operation by calling vbblk_init.

Calling vbblk_done with type as CS_BLK_BATCH marks a complete batch in a bulk copy in operation. CS_BLK_BATCH is legal only for bulk copy in operations.

Calling vbblk_done with type as CS_BLK_CANCEL cancels the current bulk copy operation. Rows transferred since an application's last vbblk_done(CS_BLK_BATCH) call are not saved in the database. Once an application cancels a bulk copy operation, it cannot call any bulk copy routines (except for vbblk_drop and vbblk_alloc) until it initializes a new bulk copy operation by calling vbblk_init.

Bulk Copies Into the Database

When an application bulk copies data into a database, the rows are permanently saved only when the application calls vbblk_done.   If an application has large amounts of data to transfer, it can be convenient to "batch" the data into smaller units of recoverability.

An application can batch rows by calling vbblk_done with type as CS_BLK_BATCH once every n rows or when there is a lull between periods of data, as in a telemetry application.   This causes all rows transferred since the application's last vbblk_done call to be permanently saved.

After saving a batch of rows, an application's first call to vbblk_rowxfer implicitly starts the next batch.

An application must call vbblk_done with type as CS_BLK_ALL to send its final batch of rows.   This call permanently saves the rows, marks the end of the bulk copy operation, and cleans up internal bulk copy data structures.

Bulk Copies Out From the Database

After transferring the last row in a bulk copy out operation, an application must call vbblk_done with type as CS_BLK_ALL to mark the end of the bulk copy operation and clean up internal bulk copy data structures.

See Also

vbblk_init, vbblk_rowxfer

vbblk_get_data

Retrieve data from a bulk copy row.

Syntax

vbblk_get_data(column&)

Parameters

column&

The column number indicates the column from which the data is to be retrieved.

Results

A string representation of the data for the requested columns.

Remarks

See Also

vbblk_init

Initiate a bulk copy operation.

Syntax

vbblk_init(blkdesc&, direction&, tablename$)

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer. This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

direction&

The direction of the bulk copy. The following table lists the symbolic values that are legal for direction:

Value of direction:	**vbblk_init:**
CS_BLK_IN	Initializes a bulk copy operation into the database.
CS_BLK_OUT	Initializes a bulk copy operation out from the database.

tablename$

A pointer to the name of the table of interest. Any legal server table name is acceptable.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. A common cause of failure is specifying a non-existent table.
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.

Remarks

vbblk_init is both a CT-Library and Server-Library bulk copy routine.

When a bulk copy operation is complete, an application must call vbblk_done with type as CS_BLK_ALL in order to mark the end of the bulk copy operation and clean up internal bulk copy data structures.

See Also

vbblk_bind, vbblk_done, vbblk_rowxfer

vbblk_props_num

Set or retrieve bulk descriptor structure properties.

Syntax

vbblk_props_num(blkdesc&, action&, property& buffer&, outlen&)

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer.   This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

action&

One of the following symbolic values:

Value of action:	**vbblk_props_num:**
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer&

If a property value is being set, buffer points to the value to use in setting the property.

If a property value is being retrieved, buffer points to the space in which vbblk_props_num will place the requested information.

outlen&

A pointer to an integer variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbblk_props_num sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed

Remarks

Bulk descriptor properties define aspects of a specific bulk copy operation.

An application must set bulk copy properties after calling vbblk_alloc to allocate a bulk descriptor structure and before calling vbblk_init to initiate a specific bulk copy operation.

An application can use vbblk_props_num to set or retrieve the following properties:

Property name:	**What it is:**	**buffer is:**	**Applicable to?**	**Notes**
BLK_IDENTITY	Whether a table's identity column is included in the bulk copy operation.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	IN copies only	
BLK_SENSITIVITY_LBL	Whether a table's sensitivity column is included in the bulk copy operation.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	Both IN and OUT copies	Secure SQL Server only

About the Properties

BLK_IDENTITY

- BLK_IDENTITY determines whether a table's identity column is included in a bulk copy in operation.

- BLK_IDENTITY does not affect bulk copy out operations.

- If BLK_IDENTITY is CS_TRUE, the application must supply data for the identity column.

If BLK_IDENTITY is CS_FALSE, the application does not need to supply data for the identity column. In this case, the server supplies a default value for the column.

- BLK_IDENTITY works by setting identity_insert on for the database table of interest. This allows values to be inserted into the identity column. When the bulk copy operation is finished, the identity_insert option for the table is reset to off.

BLK_SENSITIVITY_LBL

- BLK_SENSITIVITY_LBL determines whether or not data for the sensitivity column is included in a bulk copy operation. By default, sensitivity column data is not included.

- BLK_SENSITIVITY_LBL affects both bulk copy in and bulk copy out operations.

- If BLK_SENSITIVITY_LBL is CS_TRUE, the application must supply data for the sensitivity column on bulk copy in operations andwill receive data from the sensitivity column on bulk copy out operations.

If BLK_SENSITIVITY_LBL is CS_FALSE, the application does not need to supply data for the sensitivity column on bulk copy in operations and will not receive data from the sensitivity column on bulk copy out operations.

- BLK_SENSITIVITY_LBL is applicable to Secure SQL Server copies only. vbblk_init fails if BLK_SENSITIVITY_LBL is CS_TRUE and a bulk copy operation is attempted against a standard SQL Server.

- Users copying into the sensitivity column must have the bcpin_labels_role role activated on Secure SQL Server. vbblk_init fails if the user does not have the bcpin_labels_role.

See Also

vbblk_alloc, vbblk_init

vbblk_props_str

Set or retrieve bulk descriptor structure properties.

Syntax

vbblk_props_str(blkdesc&, action&, property& buffer$, outlen&)

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer.   This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

action&

One of the following symbolic values:

Value of action:	vbblk_props_str:
CS_SET	Sets the value of the property.
CS_GET	Retrieves the value of the property.
CS_CLEAR	Clears the value of the property by resetting it to its default value.

property&

The symbolic name of the property whose value is being set or retrieved.

buffer$

If a property value is being set, buffer points to the value to use in setting the property.

If a property value is being retrieved, buffer points to the space in which vbblk_props_str will place the requested information.

outlen&

A pointer to an integer variable.

outlen is not used if a property value is being set and should be passed as NULL.

If a property value is being retrieved and outlen is supplied, vbblk_props_str sets outlen to the length, in bytes, of the requested information.

If the information is larger than buflen bytes, an application can use the value of outlen to determine how many bytes are needed to hold the information.

Results

Returns:	To Indicate:
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed

Remarks

Bulk descriptor properties define aspects of a specific bulk copy operation.

An application must set bulk copy properties after calling vbblk_alloc to allocate a bulk descriptor structure and before calling vbblk_init to initiate a specific bulk copy operation.

An application can use vbblk_props_str to set or retrieve the following properties:

Property name:	What it is:	buffer is:	Applicable to?	Notes
BLK_IDENTITY	Whether a table's identity column is included in the bulk copy operation.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	IN copies only	
BLK_SENSITIVITY_LBL	Whether a table's sensitivity column is included in the bulk copy operation.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	Both IN and OUT copies	Secure SQL Server only

About the Properties

BLK_IDENTITY

- BLK_IDENTITY determines whether a table's identity column is included in a bulk copy in operation.

- BLK_IDENTITY does not affect bulk copy out operations.

- If BLK_IDENTITY is CS_TRUE, the application must supply data for the identity column.

If BLK_IDENTITY is CS_FALSE, the application does not need to supply data for the identity column. In this case, the server supplies a default value for the column.

- BLK_IDENTITY works by setting identity_insert on for the database table of interest. This allows values to be inserted into the identity column. When the bulk copy operation is finished, the identity_insert option for the table is reset to off.

BLK_SENSITIVITY_LBL

- BLK_SENSITIVITY_LBL determines whether or not data for the sensitivity column is included in a bulk copy operation. By default, sensitivity column data is not included.

- BLK_SENSITIVITY_LBL affects both bulk copy in and bulk copy out operations.

- If BLK_SENSITIVITY_LBL is CS_TRUE, the application must supply data for the sensitivity column on bulk copy in operations andwill receive data from the sensitivity column on bulk copy out operations.

If BLK_SENSITIVITY_LBL is CS_FALSE, the application does not need to supply data for the sensitivity column on bulk copy in operations and will not receive data from the sensitivity column on bulk copy out operations.

- BLK_SENSITIVITY_LBL is applicable to Secure SQL Server copies only. vbblk_init fails if BLK_SENSITIVITY_LBL is CS_TRUE and a bulk copy operation is attempted against a standard SQL Server.

- Users copying into the sensitivity column must have the bcpin_labels_role role activated on Secure SQL Server. vbblk_init fails if the user does not have the bcpin_labels_role.

See Also

vbblk_alloc, vbblk_init

vbblk_set_data

Sets data into a column for a bulk copy row.

Syntax

vbblk_set_data(dat$, column&)

Parameters

dat$

A string representation of the data to be placed in a column.

column&

The column number indicates the column from which the data is to be stored.

Results

Remarks

See Also

vbblk_rowxfer

Transfer one or more rows during a bulk copy operation.

Syntax

vbblk_rowxfer(blkdesc&)

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer. This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_PENDING	Asynchronous network I/O is in effect. For more information, refer to Asychronous Programming.
CS_BLK_HAS_TEXT	The row contains one or more columns which have been marked for transfer via vbblk_textxfer. The application must call vbblk_textxfer to transfer data for these columns row before calling vbblk_rowxfer to transfer the next row.
CS_END_DATA	When copying data out from a database, vbblk_rowxfer returns CS_END_DATA to indicate that all rows have been transferred. When copying data into a database, vbblk_rowxfer does not return CS_END_DATA.

A common reason for a vbblk_rowxfer failure is conversion error.

Remarks

vbblk_rowxfer is a CT-Library bulk copy routine.

The number of rows transferred per vbblk_rowxfer call is determined by the value of datafmt.count in an application's vbblk_bind calls for the operation. If datafmt.count is greater than 1, array binding is considered to be in effect and multiple rows are transferred per vbblk_rowxfer call.

An application calls vbblk_rowxfer to do the following:

- Transfer rows into the database, during a bulk copy in operation.
- Transfer rows out from the database, during a bulk copy out operation.

If array binding is not in effect, an application can use vbblk_textxfer in conjunction with vbblk_rowxfer to transfer rows containing large text or image values.

When transferring rows that contain large text or image columns, it is often convenient for an application to transfer the text or image data in fixed-size chunks, rather than all at once. Complete the following to accomplish this:

- The application passes buffer as NULL in its vbblk_bind call for the column. This tells CT-Library that data for this column will be transferred using vbblk_textxfer.
- When the application calls vbblk_rowxfer to transfer the row, vbblk_rowxfer returns CS_BLK_HAS_TEXT, indicating that CT-Library expects further data for this row to be transferred with vbblk_textxfer.
- Then, for each column requiring transfer, the application calls vbblk_textxfer in a loop that terminates when vbblk_textxfer returns CS_END_DATA, indicating that all of the data for this column has been transferred.

A bulk copy operation is not automatically terminated if vbblk_rowxfer returns CS_FAIL. An application can continue to call vbblk_rowxfer after correcting (copy in operations only) or discarding the problem row.

See Also

vbblk_bind, vbblk_textxfer

vbblk_textxfer

Transfer a column's data in chunks during a bulk copy operation.

Syntax

vbblk_textxfer(blkdesc&, buffer$, outlen&)

Parameters

blkdesc&

The parameter is primed with the bulk copy description pointer. This bulk copy description pointer is a parameter used in functions such as vbblk_bind.

buffer$

A pointer to the space from which vbblk_textxfer picks up the chunk of text, image, sensitivity, or boundary data.

outlen&

A pointer to an integer variable.

outlen is not used for a bulk copy in operation and should be passed as NULL.

For a bulk copy out operation, outlen represents the length, in bytes, of the data copied to buffer.

Results

Returns:	**To Indicate:**
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_END_DATA	When copying data out from a database, vbblk_textxfer returns CS_END_DATA to indicate that a complete column value has been sent. When copying data into a database, vbblk_textxfer returns CS_END_DATA when an amount of data equal to vbblk_bind's datalen has been sent.
CS_PENDING	Asynchronous network I/O is in effect. Refer to Asychronous Programming.

Remarks

vbblk_textxfer is a CT-Library bulk copy routine.

Although vbblk_textxfer is primarily useful for transferring large text or image values, it can be used to transfer data of any datatype. However, vbblk_textxfer does not perform any data conversion, but simply transfers data.

There are two ways for an application to transfer text and image values during a bulk copy operation:

- The application can treat text or image data like ordinary data: For example, it can bind columns to program variables and transfer rows using vbblk_rowxfer. Usually, this method is convenient for small text and image values but not larger ones because it requires an application to allocate program variables large enough to hold entire columns.

- The application can transfer text or image data in chunks, using vbblk_textxfer.

An application marks a column for transfer via vbblk_textxfer by calling vbblk_bind for the column with a NULL buffer parameter.

Bulk Copies Into a Database

An application's vbblk_bind calls do not have to be in column order, but data for vbblk_textxfer columns must be transferred in column order.

For example, an application can bind columns 3 and 4, and then mark columns 2 and 1 for transfer via blk_textxfer.   After calling vbblk_rowxfer to copy data for columns 3 and 4, the application needs to call vbblk_textxfer to transfer data for column 1 before calling it for column 2.

When copying data into a database, once a text, image, boundary, or sensitivity column is marked for transfer via vbblk_textxfer, all subsequent columns of these types must also be marked for transfer via vbblk_textxfer.

This means an application cannot mark the first text column in a row for transfer via vbblk_textxfer and then bind a subsequent text column to a program variable.

When copying data into a database, an application is responsible for calling vbblk_textxfer the correct number of times to transfer the complete text or image value.

Bulk Copies From a Database

When using vbblk_textxfer to copy data out of a database, only columns that follow bound columns are available for transfer via vbblk_textxfer.

This means an application cannot bind the first two columns in a row to program variables, mark the third for transfer via vbblk_textxfer, and bind the fourth.   Another way to think of it is that columns being transferred via vbblk_textxfer must reside at the end of row.

When copying data out from a database, vbblk_textxfer returns CS_END_DATA to indicate that a complete column value has been copied.

See Also

vbblk_bind, vbblk_rowxfer