PUBLIC
SQL Anywhere Server
Document Version: 17.01.0 – 2021-10-15

SQL Anywhere - C API Reference

THE BEST RUN SAP

Content

1 SQL Anywhere C API Reference

The specific C API elements are defined in header files.

Header Files

- `sacapi.h`
- `sacapidll.h`

Remarks

The `sacapi.h` header file defines the SQL Anywhere C API entry points.

The `sacapidll.h` header file defines the C API library initialization and finalization functions. You must include `sacapidll.h` in your source files and include the source code from `sacapidll.c`.

The SQL Anywhere C API reference is available in the *SQL Anywhere- C API Reference* at https://help.sap.com/viewer/b86b137c54474b11b955a4d16358b208/LATEST/en-US.

In this section:

Version 2 introduced the "_ex" functions and the ability to cancel requests.

SQLANY_API_VERSION_3 Variable [page 67]
Version 3 introduced the "callback" function.

SQLANY_API_VERSION_4 Variable [page 67]
Version 4 introduced NCHAR support and wide inserts.

SQLANY_CALLBACK Variable [page 68]
Callback function type.

1.1 sqlany_affected_rows(a_sqlany_stmt *) Method

Returns the number of rows affected by execution of the prepared statement.

Syntax

```
public sacapi_i32 sqlany_affected_rows (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement that was prepared and executed successfully with no result set returned. For example, an INSERT, UPDATE or DELETE statement was executed.

Returns

The number of rows affected or -1 on failure.

Related Information

sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]

1.2 sqlany_bind_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method

Binds a user-supplied buffer as a result set column to the prepared statement.

Syntax

```
public sacapi_bool sqlany_bind_column (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    a_sqlany_data_value * value
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
index The index of the column. This number must be between 0 and sqlany_num_cols() - 1.
value An a_sqlany_data_value structure describing the bound buffers, or NULL to clear previous binding information.

Returns

1 on success or 0 on unsuccessful.

Remarks

If the size of the fetched row set is greater than 1, the buffer must be large enough to hold the data of all of the rows in the row set. This function can also be used to clear the binding of a column by specifying value to be NULL.

Related Information

sqlany_clear_column_bindings(a_sqlany_stmt *) Method [page 11]
sqlany_set_rowset_size(a_sqlany_stmt *, sacapi_u32) Method [page 50]

1.3 sqlany_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method

Bind a user-supplied buffer as a parameter to the prepared statement.

Syntax

```
public sacapi_bool sqlany_bind_param (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    a_sqlany_bind_param * param
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
index The index of the parameter. This number must be between 0 and sqlany_num_params() - 1.
param An a_sqlany_bind_param structure description of the parameter to be bound.

Returns

1 on success or 0 on unsuccessful.

Related Information

sqlany_describe_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 15]

1.4 sqlany_cancel(a_sqlany_connection *) Method

Cancel an outstanding request on a connection.

Syntax

```
public void sqlany_cancel (a_sqlany_connection * sqlany_conn)
```

Parameters

sqlany_conn A connection object with a connection established using sqlany_connect().

1.5 sqlany_clear_column_bindings(a_sqlany_stmt *) Method

Removes all column bindings defined using sqlany_bind_column().

Syntax

```
public sacapi_bool sqlany_clear_column_bindings (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().

Returns

1 on success or 0 on failure.

Related Information

sqlany_bind_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method [page 9]

1.6 sqlany_clear_error(a_sqlany_connection *) Method

Clears the last stored error code.

Syntax

```
public void sqlany_clear_error (a_sqlany_connection * sqlany_conn)
```

Parameters

sqlany_conn A connection object returned from sqlany_new_connection().

Related Information

sqlany_new_connection(void) Method [page 38]

1.7 sqlany_client_version(char *, size_t) Method

Returns the current client version.

Syntax

```
public sacapi_bool sqlany_client_version (
    char * buffer,
    size_t len
)
```

Parameters

buffer The buffer to be filled with the client version string.
len The length of the buffer supplied.

Returns

1 when successful or 0 when unsuccessful.

Remarks

This method fills the buffer passed with the major, minor, patch, and build number of the client library. The buffer will be null-terminated.

1.8 sqlany_client_version_ex(a_sqlany_interface_context *, char *, size_t) Method

Returns the current client version.

Syntax

```
public sacapi_bool sqlany_client_version_ex (
    a_sqlany_interface_context * context,
    char * buffer,
    size_t len
)
```

Parameters

context object that was create with sqlany_init_ex()
buffer The buffer to be filled with the client version string.
len The length of the buffer supplied.

Returns

1 when successful or 0 when unsuccessful.

Remarks

This method fills the buffer passed with the major, minor, patch, and build number of the client library. The buffer will be null-terminated.

Related Information

sqlany_init_ex(const char *, sacapi_u32, sacapi_u32 *) Method [page 35]

1.9 sqlany_commit(a_sqlany_connection *) Method

Commits the current transaction.

Syntax

```
public sacapi_bool sqlany_commit (a_sqlany_connection * sqlany_conn)
```

Parameters

sqlany_conn The connection object on which the commit operation is performed.

Returns

1 when successful or 0 when unsuccessful.

Related Information

sqlany_rollback(a_sqlany_connection *) Method [page 45]

1.10 sqlany_connect(a_sqlany_connection *, const char *) Method

Creates a connection to a SQL Anywhere database server using the supplied connection object and connection string.

Syntax

```
public sacapi_bool sqlany_connect (
    a_sqlany_connection * sqlany_conn,
    const char * str
)
```

Parameters

sqlany_conn A connection object created by sqlany_new_connection().
str A SQL Anywhere connection string.

Returns

1 if the connection is established successfully or 0 when the connection fails. Use sqlany_error() to retrieve the error code and message.

Remarks

The supplied connection object must first be allocated using sqlany_new_connection().

The following example demonstrates how to retrieve the error code of a failed connection attempt:

```
a_sqlany_connection * sqlany_conn;
sqlany_conn = sqlany_new_connection();
if( !sqlany_connect( sqlany_conn, "uid=dba;pwd=passwd" ) ) {
   char reason[SACAPI_ERROR_SIZE];
   sacapi_i32 code;
   code = sqlany_error( sqlany_conn, reason, sizeof(reason) );
   printf( "Connection failed. Code: %d Reason: %s\n", code, reason );
} else {
   printf( "Connected successfully!\n" );
   sqlany_disconnect( sqlany_conn );
}
sqlany_free_connection( sqlany_conn );
```

Related Information

sqlany_new_connection(void) Method [page 38]
sqlany_error(a_sqlany_connection *, char *, size_t) Method [page 17]

1.11 sqlany_describe_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method

Describes the bind parameters of a prepared statement.

Syntax

```
public sacapi_bool sqlany_describe_bind_param (
```

```
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    a_sqlany_bind_param * param
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
index The index of the parameter. This number must be between 0 and sqlany_num_params() - 1.
param An a_sqlany_bind_param structure that is populated with information.

Returns

1 when successful or 0 when unsuccessful.

Remarks

This function allows the caller to determine information about prepared statement parameters. The type of prepared statement, stored procedure, or DML statement determines the amount of information provided. The direction of the parameters (input, output, or input-output) are always provided.

Related Information

sqlany_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 10]
sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.12 sqlany_disconnect(a_sqlany_connection *) Method

Disconnects an already established SQL Anywhere connection.

Syntax

```
public sacapi_bool sqlany_disconnect (a_sqlany_connection * sqlany_conn)
```

Parameters

sqlany_conn A connection object with a connection established using sqlany_connect().

Returns

1 when successful or 0 when unsuccessful.

Remarks

All uncommitted transactions are rolled back.

Related Information

sqlany_connect(a_sqlany_connection *, const char *) Method [page 14]
sqlany_new_connection(void) Method [page 38]

1.13 sqlany_error(a_sqlany_connection *, char *, size_t) Method

Retrieves the last error code and message stored in the connection object.

Syntax

```
public sacapi_i32 sqlany_error (
    a_sqlany_connection * sqlany_conn,
    char * buffer,
    size_t size
)
```

Parameters

sqlany_conn A connection object returned from sqlany_new_connection().
buffer A buffer to be filled with the error message.
size The size of the supplied buffer.

Returns

The last error code. Positive values are warnings, negative values are errors, and 0 indicates success.

Related Information

sqlany_connect(a_sqlany_connection *, const char *) Method [page 14]

1.14 sqlany_execute(a_sqlany_stmt *) Method

Executes a prepared statement.

Syntax

```
public sacapi_bool sqlany_execute (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().

Returns

1 if the statement is executed successfully or 0 on failure.

Remarks

You can use sqlany_num_cols() to verify if the executed statement returned a result set.

The following example shows how to execute a statement that does not return a result set:

```
a_sqlany_stmt *          stmt;
int                      i;
a_sqlany_bind_param   param;
stmt = sqlany_prepare( sqlany_conn, "insert into moe(id,value) values( ?,? )" );
if( stmt ) {
   sqlany_describe_bind_param( stmt, 0, &param );
   param.value.buffer = (char *)&i;
   param.value.type   = A_VAL32;
```

```
    sqlany_bind_param( stmt, 0, &param );
    sqlany_describe_bind_param( stmt, 1, &param );
    param.value.buffer = (char *)&i;
    param.value.type   = A_VAL32;
    sqlany_bind_param( stmt, 1, &param );
    for( i = 0; i < 10; i++ ) {
        if( !sqlany_execute( stmt ) ) {
                // call sqlany_error()
            }
    }
    sqlany_free_stmt( stmt );
}
```

Related Information

sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.15 sqlany_execute_direct(a_sqlany_connection *, const char *) Method

Executes the SQL statement specified by the string argument and possibly returns a result set.

Syntax

```
public a_sqlany_stmt * sqlany_execute_direct (
    a_sqlany_connection * sqlany_conn,
    const char * sql_str
)
```

Parameters

sqlany_conn A connection object with a connection established using sqlany_connect().
sql_str A SQL string. The SQL string should not have parameters such as ?.

Returns

A statement handle if the function executes successfully, NULL when the function executes unsuccessfully.

Remarks

Use this method to prepare and execute a statement, or instead of calling sqlany_prepare() followed by sqlany_execute().

The following example shows how to execute a statement that returns a result set:

```
a_sqlany_stmt *   stmt;
stmt = sqlany_execute_direct( sqlany_conn, "select * from employees" );
if( stmt && sqlany_num_cols( stmt ) > 0 ) {
   while( sqlany_fetch_next( stmt ) ) {
       int i;
           for( i = 0; i < sqlany_num_cols( stmt ); i++ ) {
           // Get column i data
       }
   }
   sqlany_free_stmt( stmt  );
}
```

> **Note**
>
> This function cannot be used for executing a SQL statement with parameters.

Related Information

sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]
sqlany_num_cols(a_sqlany_stmt *) Method [page 40]
sqlany_get_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method [page 28]

1.16 sqlany_execute_immediate(a_sqlany_connection *, const char *) Method

Executes the supplied SQL statement immediately without returning a result set.

> **Syntax**

```
public sacapi_bool sqlany_execute_immediate (
    a_sqlany_connection * sqlany_conn,
    const char * sql
)
```

Parameters

sqlany_conn A connection object with a connection established using sqlany_connect().
sql A string representing the SQL statement to be executed.

Returns

1 on success or 0 on failure.

Remarks

This function is useful for SQL statements that do not return a result set.

1.17 sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method

Moves the current row in the result set to the specified row number and then fetches rows of data starting from the current row.

Syntax

```
public sacapi_bool sqlany_fetch_absolute (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_i32 row_num
)
```

Parameters

sqlany_stmt A statement object that was executed by sqlany_execute() or sqlany_execute_direct().
row_num The row number to be fetched. The first row is 1, the last row is -1.

Returns

1 if the fetch was successfully, 0 when the fetch is unsuccessful.

Remarks

The number of rows fetched is set using the sqlany_set_rowset_size() function. By default, one row is returned.

Related Information

sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_error(a_sqlany_connection *, char *, size_t) Method [page 17]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]
sqlany_set_rowset_size(a_sqlany_stmt *, sacapi_u32) Method [page 50]

1.18 sqlany_fetch_next(a_sqlany_stmt *) Method

Returns the next set of rows from the result set.

Syntax

```
public sacapi_bool sqlany_fetch_next (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement object that was executed by sqlany_execute() or sqlany_execute_direct().

Returns

1 if the fetch was successfully, 0 when the fetch is unsuccessful.

Remarks

When the result object is first created, the current row pointer is set to before the first row, that is, row 0. This function first advances the row pointer to the next unfetched row and then fetches rows of data starting from that row. The number of rows fetched is set by the sqlany_set_rowset_size() function. By default, one row is returned.

Related Information

sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_error(a_sqlany_connection *, char *, size_t) Method [page 17]
sqlany_set_rowset_size(a_sqlany_stmt *, sacapi_u32) Method [page 50]

1.19 sqlany_fetched_rows(a_sqlany_stmt *) Method

Returns the number of rows fetched.

Syntax

```
public sacapi_i32 sqlany_fetched_rows (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().

Returns

The number of rows fetched or -1 on failure.

Remarks

In general, the number of rows fetched is equal to the size specified by the sqlany_set_rowset_size() function. The exception is when there are fewer rows from the fetch position to the end of the result set than specified, in which case the number of rows fetched is smaller than the specified row set size. The function returns -1 if the last fetch was unsuccessful or if the statement has not been executed. The function returns 0 if the statement has been executed but no fetching has been done.

Related Information

sqlany_bind_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method [page 9]

sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]
sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]

1.20 sqlany_finalize_interface(SQLAnywhereInterface *) Method

Unloads the C API DLL library and resets the SQLAnywhereInterface structure.

Syntax

```
public void sqlany_finalize_interface (SQLAnywhereInterface * api)
```

Parameters

api An initialized structure to finalize.

Remarks

Use the following statement to include the function prototype:

```
#include "sacapidll.h"
```

Use this method to finalize and free resources associated with the SQL Anywhere C API DLL.

Examples of how the sqlany_finalize_interface method is used can be found in the C API examples in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

1.21 sqlany_fini() Method

Finalizes the interface.

Syntax

```
public void sqlany_fini ()
```

Remarks

Frees any resources allocated by the API.

Related Information

sqlany_init(const char *, sacapi_u32, sacapi_u32 *) Method [page 34]

1.22 sqlany_fini_ex(a_sqlany_interface_context *) Method

Finalize the interface that was created using the specified context.

Syntax

```
public void sqlany_fini_ex (a_sqlany_interface_context * context)
```

Parameters

context A context object that was returned from sqlany_init_ex()

Related Information

sqlany_init_ex(const char *, sacapi_u32, sacapi_u32 *) Method [page 35]

1.23 sqlany_free_connection(a_sqlany_connection *) Method

Frees the resources associated with a connection object.

Syntax

```
public void sqlany_free_connection (a_sqlany_connection * sqlany_conn)
```

Parameters

sqlany_conn A connection object created with sqlany_new_connection().

Related Information

sqlany_new_connection(void) Method [page 38]

1.24 sqlany_free_stmt(a_sqlany_stmt *) Method

Frees resources associated with a prepared statement object.

Syntax

```
public void sqlany_free_stmt (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement object returned by the successful execution of sqlany_prepare() or sqlany_execute_direct().

Related Information

sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]

1.25 sqlany_get_batch_size(a_sqlany_stmt *) Method

Retrieves the size of the row array for a batch execute.

Syntax

```
public sacapi_u32 sqlany_get_batch_size (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().

Returns

The size of the row array.

Related Information

sqlany_set_batch_size(a_sqlany_stmt *, sacapi_u32) Method [page 46]

1.26 sqlany_get_bind_param_info(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param_info *) Method

Retrieves information about the parameters that were bound using sqlany_bind_param().

Syntax

```
public sacapi_bool sqlany_get_bind_param_info (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    a_sqlany_bind_param_info * info
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
index The index of the parameter. This number should be between 0 and sqlany_num_params() - 1.
info A sqlany_bind_param_info buffer to be populated with the bound parameter's information.

Returns

1 on success or 0 on failure.

Related Information

sqlany_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 10]
sqlany_describe_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 15]
sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.27 sqlany_get_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method

Fills the supplied buffer with the value fetched for the specified column at the current row.

Syntax

```
public sacapi_bool sqlany_get_column (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    a_sqlany_data_value * buffer
)
```

Parameters

sqlany_stmt A statement object executed by sqlany_execute() or sqlany_execute_direct().
col_index The number of the column to be retrieved. The column number is between 0 and sqlany_num_cols() - 1.
buffer An a_sqlany_data_value object to be filled with the data fetched for column col_index at the current row in the row set.

Returns

1 on success or 0 for failure. A failure can happen if any of the parameters are invalid or if there is not enough memory to retrieve the full value from the SQL Anywhere database server.

Remarks

When a sqlany_fetch_absolute() or sqlany_fetch_next() function is executed, a row set is created and the current row is set to be the first row in the row set. The current row is set using the sqlany_set_rowset_pos() function.

For A_BINARY and A_STRING * data types, value->buffer points to an internal buffer associated with the result set. Do not rely upon or alter the content of the pointer buffer as it changes when a new row is fetched or when the result set object is freed. Users should copy the data out of those pointers into their own buffers.

The value->length field indicates the number of valid characters that value->buffer points to. The data returned in value->buffer is not null-terminated. This function fetches all the returned values from the SQL Anywhere database server. For example, if the column contains a blob, this function attempts to allocate enough memory to hold that value. If you do not want to allocate memory, use sqlany_get_data() instead.

Related Information

sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]
sqlany_set_rowset_pos(a_sqlany_stmt *, sacapi_u32) Method [page 49]

1.28 sqlany_get_column_info(a_sqlany_stmt *, sacapi_u32, a_sqlany_column_info *) Method

Retrieves column metadata information and fills the a_sqlany_column_info structure with information about the column.

Syntax

```
public sacapi_bool sqlany_get_column_info (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    a_sqlany_column_info * buffer
)
```

Parameters

sqlany_stmt A statement object created by sqlany_prepare() or sqlany_execute_direct().
col_index The column number between 0 and sqlany_num_cols() - 1.
buffer A column info structure to be filled with column information.

Returns

1 on success or 0 if the column index is out of range, or if the statement does not return a result set.

Related Information

sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.29 sqlany_get_data(a_sqlany_stmt *, sacapi_u32, size_t, void *, size_t) Method

Retrieves the data fetched for the specified column at the current row into the supplied buffer memory.

Syntax

```
public sacapi_i32 sqlany_get_data (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    size_t offset,
    void * buffer,
    size_t size
)
```

Parameters

sqlany_stmt A statement object executed by sqlany_execute() or sqlany_execute_direct().
col_index The number of the column to be retrieved. The column number is between 0 and sqlany_num_cols() - 1.
offset The starting offset of the data to get.
buffer A buffer to be filled with the contents of the column at the current row in the row set. The buffer pointer must be aligned correctly for the data type copied into it.
size The size of the buffer in bytes. The function fails if you specify a size greater than 2^31 - 1.

Returns

The number of bytes successfully copied into the supplied buffer. This number must not exceed 2^31 - 1. 0 indicates that no data remains to be copied. -1 indicates a failure.

Remarks

When a sqlany_fetch_absolute() or sqlany_fetch_next() function is executed, a row set is created and the current row is set to be the first row in the row set. The current row is set using the sqlany_set_rowset_pos() function.

Related Information

sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]
sqlany_set_rowset_pos(a_sqlany_stmt *, sacapi_u32) Method [page 49]

1.30 sqlany_get_data_info(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_info *) Method

Retrieves information about the fetched data at the current row.

Syntax

```
public sacapi_bool sqlany_get_data_info (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 col_index,
    a_sqlany_data_info * buffer
)
```

Parameters

sqlany_stmt A statement object executed by sqlany_execute() or sqlany_execute_direct().
col_index The column number between 0 and sqlany_num_cols() - 1.
buffer A data info buffer to be filled with the metadata about the data at the current row in the row set.

Returns

1 on success, and 0 on failure. Failure is returned when any of the supplied parameters are invalid.

Remarks

When a sqlany_fetch_absolute() or sqlany_fetch_next() function is executed, a row set is created and the current row is set to be the first row in the row set. The current row is set using the sqlany_set_rowset_pos() function.

Related Information

sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]
sqlany_set_rowset_pos(a_sqlany_stmt *, sacapi_u32) Method [page 49]

1.31 sqlany_get_next_result(a_sqlany_stmt *) Method

Advances to the next result set in a multiple result set query.

Syntax

```
public sacapi_bool sqlany_get_next_result (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement object executed by sqlany_execute() or sqlany_execute_direct().

Returns

1 if the statement successfully advances to the next result set, 0 otherwise.

Remarks

If a query (such as a call to a stored procedure) returns multiple result sets, then this function advances from the current result set to the next.

The following example demonstrates how to advance to the next result set in a multiple result set query:

```
stmt = sqlany_execute_direct( sqlany_conn, "call
my_multiple_results_procedure()" );
if( result ) {
   do {
       while( sqlany_fetch_next( stmt ) ) {
          // get column data
       }
   } while( sqlany_get_next_result( stmt ) );
   sqlany_free_stmt( stmt );
}
```

Related Information

sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_execute(a_sqlany_stmt *) Method [page 18]

1.32 sqlany_get_rowset_size(a_sqlany_stmt *) Method

Retrieves the size of the row set to be fetched by the sqlany_fetch_absolute() and sqlany_fetch_next() functions.

Syntax

```
public sacapi_u32 sqlany_get_rowset_size (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().

Returns

The size of the row set, or 0 if the statement does not return a result set.

Related Information

sqlany_set_rowset_size(a_sqlany_stmt *, sacapi_u32) Method [page 50]

sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]

1.33 sqlany_init(const char *, sacapi_u32, sacapi_u32 *) Method

Initializes the interface.

Syntax

```
public sacapi_bool sqlany_init (
    const char * app_name,
    sacapi_u32 api_version,
    sacapi_u32 * version_available
)
```

Parameters

app_name A string that names the application that is using the API. For example, "PHP", "PERL", or "RUBY".

api_version The version of the compiled application.

version_available An optional argument to return the maximum supported API version.

Returns

1 on success, 0 otherwise

Remarks

The following example demonstrates how to initialize the SQL Anywhere C API DLL:

```
sacapi_u32 api_version;
if( sqlany_init( "PHP", SQLANY_API_VERSION_1, &api_version ) ) {
   printf( "Interface initialized successfully!\n" );
} else {
   printf( "Failed to initialize the interface! Supported version=%d\n",
api_version );
}
```

Related Information

sqlany_fini() Method [page 24]

1.34 sqlany_init_ex(const char *, sacapi_u32, sacapi_u32 *) Method

Initializes the interface using a context.

Syntax

```
public a_sqlany_interface_context * sqlany_init_ex (
    const char * app_name,
    sacapi_u32 api_version,
    sacapi_u32 * version_available
)
```

Parameters

app_name A string that names the API used, for example "PHP", "PERL", or "RUBY".
api_version The current API version that the application is using. This should normally be one of the SQLANY_API_VERSION_* macros
version_available An optional argument to return the maximum API version that is supported.

Returns

a context object on success and NULL on failure.

Related Information

sqlany_fini_ex(a_sqlany_interface_context *) Method [page 25]

1.35 sqlany_initialize_interface(SQLAnywhereInterface *, const char *) Method

Initializes the SQLAnywhereInterface object and loads the DLL dynamically.

Syntax

```
public int sqlany_initialize_interface (
    SQLAnywhereInterface * api,
    const char * optional_path_to_dll
)
```

Parameters

api An API structure to initialize.
optional_path_to_dll An optional argument that specifies a path to the SQL Anywhere C API DLL.

Returns

1 on successful initialization, and 0 on failure.

Remarks

Use the following statement to include the function prototype:

```
#include "sacapidll.h"
```

This function attempts to load the SQL Anywhere C API DLL dynamically and looks up all the entry points of the DLL. The fields in the SQLAnywhereInterface structure are populated to point to the corresponding functions in the DLL. If the optional path argument is NULL, the environment variable SQLANY_DLL_PATH is checked. If the variable is set, the library attempts to load the DLL specified by the environment variable. If that fails, the interface attempts to load the DLL directly (this relies on the environment being setup correctly).

Examples of how the sqlany_initialize_interface method is used can be found in the C API examples in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

1.36 sqlany_make_connection(void *) Method

Creates a connection object based on a supplied DBLIB SQLCA pointer.

Syntax

```
public a_sqlany_connection * sqlany_make_connection (void * arg)
```

Parameters

arg A void * pointer to a DBLIB SQLCA object.

Returns

A connection object.

Related Information

sqlany_new_connection(void) Method [page 38]
sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_execute_immediate(a_sqlany_connection *, const char *) Method [page 20]
sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.37 sqlany_make_connection_ex(a_sqlany_interface_context *, void *) Method

Creates a connection object based on a supplied DBLIB SQLCA pointer and context.

Syntax

```
public a_sqlany_connection * sqlany_make_connection_ex (
    a_sqlany_interface_context * context,
    void * arg
)
```

Parameters

context A valid context object that was created by sqlany_init_ex()
arg A void * pointer to a DBLIB SQLCA object.

Returns

A connection object.

Related Information

sqlany_init_ex(const char *, sacapi_u32, sacapi_u32 *) Method [page 35]
sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_execute_immediate(a_sqlany_connection *, const char *) Method [page 20]
sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.38 sqlany_new_connection(void) Method

Creates a connection object.

Syntax

```
public a_sqlany_connection * sqlany_new_connection (void)
```

Returns

A connection object

Remarks

You must create an API connection object before establishing a database connection. Errors can be retrieved from the connection object. Only one request can be processed on a connection at a time. In addition, not more than one thread is allowed to access a connection object at a time. Undefined behavior or a failure occurs when multiple threads attempt to access a connection object simultaneously.

Related Information

sqlany_connect(a_sqlany_connection *, const char *) Method [page 14]
sqlany_disconnect(a_sqlany_connection *) Method [page 16]

1.39 sqlany_new_connection_ex(a_sqlany_interface_context *) Method

Creates a connection object using a context.

Syntax

```
public a_sqlany_connection * sqlany_new_connection_ex
(a_sqlany_interface_context * context)
```

Parameters

context A context object that was returned from sqlany_init_ex()

Returns

A connection object

Related Information

sqlany_connect(a_sqlany_connection *, const char *) Method [page 14]
sqlany_disconnect(a_sqlany_connection *) Method [page 16]
sqlany_init_ex(const char *, sacapi_u32, sacapi_u32 *) Method [page 35]

1.40 sqlany_num_cols(a_sqlany_stmt *) Method

Returns number of columns in the result set.

Syntax

```
public sacapi_i32 sqlany_num_cols (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement object created by sqlany_prepare() or sqlany_execute_direct().

Returns

The number of columns in the result set or -1 on a failure.

Related Information

sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.41 sqlany_num_params(a_sqlany_stmt *) Method

Returns the number of parameters expected for a prepared statement.

Syntax

```
public sacapi_i32 sqlany_num_params (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement object returned by the successful execution of sqlany_prepare().

Returns

The expected number of parameters, or -1 if the statement object is not valid.

Related Information

sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.42 sqlany_num_rows(a_sqlany_stmt *) Method

Returns the number of rows in the result set.

Syntax

```
public sacapi_i32 sqlany_num_rows (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement object that was executed by sqlany_execute() or sqlany_execute_direct().

Returns

The number rows in the result set. If the number of rows is an estimate, the number returned is negative and the estimate is the absolute value of the returned integer. The value returned is positive if the number of rows is exact.

Remarks

By default this function only returns an estimate. To return an exact count, set the row_counts option on the connection.

Related Information

sqlany_execute_direct(a_sqlany_connection *, const char *) Method [page 19]
sqlany_execute(a_sqlany_stmt *) Method [page 18]

1.43 sqlany_prepare(a_sqlany_connection *, const char *) Method

Prepares a supplied SQL string.

Syntax

```
public a_sqlany_stmt * sqlany_prepare (
    a_sqlany_connection * sqlany_conn,
    const char * sql_str
)
```

Parameters

sqlany_conn A connection object with a connection established using sqlany_connect().
sql_str The SQL statement to be prepared.

Returns

A handle to a SQL Anywhere statement object. The statement object can be used by sqlany_execute() to execute the statement.

Remarks

Execution does not happen until sqlany_execute() is called. The returned statement object should be freed using sqlany_free_stmt().

The following statement demonstrates how to prepare a SELECT SQL string:

```
char * str;
a_sqlany_stmt * stmt;
str = "select * from employees where salary >= ?";
stmt = sqlany_prepare( sqlany_conn, str );
if( stmt == NULL ) {
   // Failed to prepare statement, call sqlany_error() for more info
```

```
}
```

Related Information

sqlany_free_stmt(a_sqlany_stmt *) Method [page 26]
sqlany_connect(a_sqlany_connection *, const char *) Method [page 14]
sqlany_execute(a_sqlany_stmt *) Method [page 18]
sqlany_num_params(a_sqlany_stmt *) Method [page 40]
sqlany_describe_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 15]
sqlany_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 10]

1.44 sqlany_register_callback(a_sqlany_connection *, a_sqlany_callback_type, SQLANY_CALLBACK_PARM) Method

Register a callback routine.

Syntax

```
public sacapi_bool sqlany_register_callback (
    a_sqlany_connection * sqlany_conn,
    a_sqlany_callback_type index,
    SQLANY_CALLBACK_PARM callback
)
```

Parameters

sqlany_conn A connection object with a connection established using sqlany_connect().
index Any of the callback types listed below.
callback Address of the callback routine.

Returns

1 when successful or 0 when unsuccessful.

Remarks

This function can be used to register callback functions.

```
A callback type can be any one of the following:
   CALLBACK_START
   CALLBACK_WAIT
   CALLBACK_FINISH
   CALLBACK_MESSAGE
   CALLBACK_CONN_DROPPED
   CALLBACK_DEBUG_MESSAGE
   CALLBACK_VALIDATE_FILE_TRANSFER
The following example shows a simple message callback routine and how to
register it.
void SQLANY_CALLBACK messages(
   a_sqlany_connection *sqlany_conn,
   a_sqlany_message_type msg_type,
   int sqlcode,
   unsigned short length,
   char *msg )
{
   size_t  mlen;
   char    mbuffer[80];
   mlen = __min( length, sizeof(mbuffer) );
   strncpy( mbuffer, msg, mlen );
   mbuffer[mlen] = '\0';
   printf( "Message is \"s".
", mbuffer );
   sqlany_sqlstate( sqlany_conn, mbuffer, sizeof( mbuffer ) );
   printf( "SQLCode(%d) SQLState(\"s")
", sqlcode, mbuffer );
}
```

1.45 sqlany_reset(a_sqlany_stmt *) Method

Resets a statement to its prepared state condition.

Syntax

```
public sacapi_bool sqlany_reset (a_sqlany_stmt * sqlany_stmt)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().

Returns

1 on success, 0 on failure.

Related Information

sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.46 sqlany_rollback(a_sqlany_connection *) Method

Rolls back the current transaction.

Syntax

```
public sacapi_bool sqlany_rollback (a_sqlany_connection * sqlany_conn)
```

Parameters

sqlany_conn The connection object on which the rollback operation is to be performed.

Returns

1 on success, 0 otherwise.

Related Information

sqlany_commit(a_sqlany_connection *) Method [page 14]

1.47 sqlany_send_param_data(a_sqlany_stmt *, sacapi_u32, char *, size_t) Method

Sends data as part of a bound parameter.

Syntax

```
public sacapi_bool sqlany_send_param_data (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 index,
    char * buffer,
    size_t size
```

```
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
index The index of the parameter. This should be a number between 0 and sqlany_num_params() - 1.
buffer The data to be sent.
size The number of bytes to send.

Returns

1 on success or 0 on failure.

Remarks

This method can be used to send a large amount of data for a bound parameter in chunks. This method can be used only when the batch size is 1.

Related Information

sqlany_prepare(a_sqlany_connection *, const char *) Method [page 42]

1.48 sqlany_set_batch_size(a_sqlany_stmt *, sacapi_u32) Method

Sets the size of the row array for a batch execute.

Syntax

```
public sacapi_bool sqlany_set_batch_size (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 num_rows
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
num_rows The number of rows for batch execution. The value must be 1 or greater.

Returns

1 on success or 0 on failure.

Remarks

The batch size is used only for an INSERT statement. The default batch size is 1. A value greater than 1 indicates a wide insert.

Related Information

sqlany_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 10]
sqlany_get_batch_size(a_sqlany_stmt *) Method [page 26]

1.49 sqlany_set_column_bind_type(a_sqlany_stmt *, sacapi_u32) Method

Sets the bind type of columns.

Syntax

```
public sacapi_bool sqlany_set_column_bind_type (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 row_size
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
row_size The byte size of the row. A value of 0 indicates column-wise binding and a positive value indicates row-wise binding.

Returns

1 on success or 0 on failure.

Remarks

The default value is 0, which indicates column-wise binding. A non-zero value indicates row-wise binding and specifies the byte size of the data structure that stores the row. The column is bound to the first element in a contiguous array of values. The address offset to the next element is computed based on the bind type:

- Column-wise binding - the byte size of the column type
- Row-wise binding - the row_size

Related Information

sqlany_bind_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method [page 9]

1.50 sqlany_set_param_bind_type(a_sqlany_stmt *, size_t) Method

Sets the bind type of parameters.

Syntax

```
public sacapi_bool sqlany_set_param_bind_type (
    a_sqlany_stmt * sqlany_stmt,
    size_t row_size
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().

row_size The byte size of the row. A value of 0 indicates column-wise binding and a positive value indicates row-wise binding.

Returns

1 on success or 0 on failure.

Remarks

The default value is 0, which indicates column-wise binding. A non-zero value indicates row-wise binding and specifies the byte size of the data structure that stores the row. The parameter is bound to the first element in a contiguous array of values. The address offset to the next element is computed based on the bind type:

- Column-wise binding - the byte size of the parameter type
- Row-wise binding - the row_size

Related Information

sqlany_bind_param(a_sqlany_stmt *, sacapi_u32, a_sqlany_bind_param *) Method [page 10]

1.51 sqlany_set_rowset_pos(a_sqlany_stmt *, sacapi_u32) Method

Sets the current row in the fetched row set.

Syntax

```
public sacapi_bool sqlany_set_rowset_pos (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 row_num
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
row_num The row number within the row set. The valid values are from 0 to sqlany_fetched_rows() - 1.

Returns

1 on success or 0 on failure.

Remarks

When a sqlany_fetch_absolute() or sqlany_fetch_next() function is executed, a row set is created and the current row is set to be the first row in the row set. The functions sqlany_get_column(), sqlany_get_data(), sqlany_get_data_info() are used to retrieve data at the current row.

Related Information

sqlany_set_rowset_size(a_sqlany_stmt *, sacapi_u32) Method [page 50]
sqlany_get_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method [page 28]
sqlany_get_data(a_sqlany_stmt *, sacapi_u32, size_t, void *, size_t) Method [page 30]
sqlany_get_data_info(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_info *) Method [page 31]
sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]

1.52 sqlany_set_rowset_size(a_sqlany_stmt *, sacapi_u32) Method

Sets the size of the row set to be fetched by the sqlany_fetch_absolute() and sqlany_fetch_next() functions.

Syntax

```
public sacapi_bool sqlany_set_rowset_size (
    a_sqlany_stmt * sqlany_stmt,
    sacapi_u32 num_rows
)
```

Parameters

sqlany_stmt A statement prepared successfully using sqlany_prepare().
num_rows The size of the row set. The value must be 1 or greater.

Returns

1 on success or 0 on failure.

Remarks

The default size of the row set is 1. Specifying num_rows to be a value greater than 1 indicates a wide fetch.

Related Information

sqlany_bind_column(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_value *) Method [page 9]
sqlany_fetch_absolute(a_sqlany_stmt *, sacapi_i32) Method [page 21]
sqlany_fetch_next(a_sqlany_stmt *) Method [page 22]
sqlany_get_rowset_size(a_sqlany_stmt *) Method [page 33]

1.53 sqlany_sqlstate(a_sqlany_connection *, char *, size_t) Method

Retrieves the current SQLSTATE.

Syntax

```
public size_t sqlany_sqlstate (
    a_sqlany_connection * sqlany_conn,
    char * buffer,
    size_t size
)
```

Parameters

sqlany_conn A connection object returned from sqlany_new_connection().
buffer A buffer to be filled with the current 5-character SQLSTATE.
size The buffer size.

Returns

The number of bytes copied into the buffer.

Related Information

sqlany_error(a_sqlany_connection *, char *, size_t) Method [page 17]

1.54 a_sqlany_callback_type Enumeration

An enumeration of the callback types.

Syntax

```
enum a_sqlany_callback_type
```

Members

Member name	Description
CALLBACK_START	This function is called just before a database request is sent to the server. CALLBACK_START is used only on Windows operating systems.
CALLBACK_WAIT	This function is called repeatedly by the interface library while the database server or client library is busy processing your database request.
CALLBACK_FINISH	This function is called after the response to a database request has been received by the DBLIB interface DLL. CALLBACK_FINISH is used only on Windows operating systems.
CALLBACK_MESSAGE	This function is called when messages are received from the server during the processing of a request. Messages can be sent to the client application from the database server using the SQL MESSAGE statement. Messages can also be generated by long running database server statements.
CALLBACK_CONN_DROPPED	This function is called when the database server is about to drop a connection because of a liveness timeout, through a DROP CONNECTION statement, or because the database server is being shut down. The connection name conn_name is passed in to allow you to distinguish between connections. If the connection was not named, it has a value of NULL.

Member name	Description
CALLBACK_DEBUG_MESSAGE	This function is called once for each debug message and is passed a null-terminated string containing the text of the debug message. A debug message is a message that is logged to the LogFile file. In order for a debug message to be passed to this callback, the LogFile connection parameter must be used. The string normally has a newline character () immediately before the terminating null character.
CALLBACK_VALIDATE_FILE_TRANSFER	This function is called when a file transfer requires validation. If the client data transfer is being requested during the execution of indirect statements such as from within a stored procedure, the client library will not allow a transfer unless the client application has registered a validation callback and the response from the callback indicates that the transfer may take place.

Remarks

The callback types correspond to the embedded SQL callback types.

Related Information

sqlany_register_callback(a_sqlany_connection *, a_sqlany_callback_type, SQLANY_CALLBACK_PARM) Method [page 43]

1.55 a_sqlany_data_direction Enumeration

A data direction enumeration.

Syntax

```
enum a_sqlany_data_direction
```

Members

Member name	Description	Value
DD_INVALID	Invalid data direction.	0x0
DD_INPUT	Input-only host variables.	0x1
DD_OUTPUT	Output-only host variables.	0x2
DD_INPUT_OUTPUT	Input and output host variables.	0x3

1.56 a_sqlany_data_type Enumeration

Specifies the data type being passed in or retrieved.

Syntax

```
enum a_sqlany_data_type
```

Members

Member name	Description
A_INVALID_TYPE	Invalid data type.
A_BINARY	Binary data. Binary data is treated as-is and no character set conversion is performed.
A_STRING	String data. The data where character set conversion is performed.
A_DOUBLE	Double data. Includes float values.
A_VAL64	64-bit integer.
A_UVAL64	64-bit unsigned integer.
A_VAL32	32-bit integer.
A_UVAL32	32-bit unsigned integer.
A_VAL16	16-bit integer.
A_UVAL16	16-bit unsigned integer.
A_VAL8	8-bit integer.
A_UVAL8	8-bit unsigned integer.

1.57 a_sqlany_message_type Enumeration

An enumeration of the message types for the MESSAGE callback.

Syntax

```
enum a_sqlany_message_type
```

Members

Member name	Description
MESSAGE_TYPE_INFO	The message type was INFO.
MESSAGE_TYPE_WARNING	The message type was WARNING.
MESSAGE_TYPE_ACTION	The message type was ACTION.
MESSAGE_TYPE_STATUS	The message type was STATUS.
MESSAGE_TYPE_PROGRESS	The message type was PROGRESS. This type of message is generated by long running database server statements such as BACKUP DATABASE and LOAD TABLE.

Related Information

sqlany_register_callback(a_sqlany_connection *, a_sqlany_callback_type, SQLANY_CALLBACK_PARM) Method [page 43]

1.58 a_sqlany_native_type Enumeration

An enumeration of the native types of values as described by the server.

Syntax

```
enum a_sqlany_native_type
```

Members

Member name	Description	Value
DT_BIGINT	64-bit signed integer.	608
DT_BINARY	Varying length binary data with a two-byte length field. When sending data, you must set the length field. The maximum length is 32767 bytes when sending data. When fetching data, the database server sets the length field. The maximum length is 32765 bytes when fetching data.	524
DT_BIT	8-bit signed integer.	624
DT_DATE	Null-terminated character string that is a valid date.	384
DT_DECIMAL	Packed decimal number (proprietary format).	484
DT_DOUBLE	8-byte floating-point number.	480
DT_FIXCHAR	Fixed-length blank-padded character string, in the CHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.	452
DT_FLOAT	4-byte floating-point number.	482
DT_INT	32-bit signed integer.	496
DT_LONGBINARY	Long binary data.	528
DT_LONGNVARCHAR	Long varying length character string, in the NCHAR character set.	640
DT_LONGVARCHAR	Long varying length character string, in the CHAR character set.	456
DT_NFIXCHAR	Fixed-length blank-padded character string, in the NCHAR character set. The maximum length, specified in bytes, is 32767. The data is not null-terminated.	632
DT_NOTYPE	No data type.	0
DT_NSTRING	Null-terminated character string, in the NCHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.	628

Member name	Description	Value
DT_NVARCHAR	Varying length character string, in the NCHAR character set, with a two-byte length field. When sending data, you must set the length field. The maximum length is 32767 bytes when sending data. When fetching data, the database server sets the length field. The maximum length is 32765 bytes when fetching data. The data is not null-terminated or blank-padded.	636
DT_SMALLINT	16-bit signed integer.	500
DT_STRING	Null-terminated character string, in the CHAR character set. The string is blank-padded if the database is initialized with blank-padded strings.	460
DT_TIME	Null-terminated character string that is a valid time.	388
DT_TIMESTAMP	Null-terminated character string that is a valid timestamp.	392
DT_TINYINT	8-bit signed integer.	604
DT_UNSBIGINT	64-bit unsigned integer.	620
DT_UNSINT	32-bit unsigned integer.	612
DT_UNSSMALLINT	16-bit unsigned integer.	616
DT_VARCHAR	Varying length character string, in the CHAR character set, with a two-byte length field. When sending data, you must set the length field. The maximum length is 32767 bytes when sending data. When fetching data, the database server sets the length field. The maximum length is 32765 bytes when fetching data. The data is not null-terminated or blank-padded.	448

Remarks

The value types correspond to the embedded SQL data types.

Related Information

sqlany_get_column_info(a_sqlany_stmt *, sacapi_u32, a_sqlany_column_info *) Method [page 29]
a_sqlany_column_info Structure [page 59]

1.59 a_sqlany_bind_param Structure

A bind parameter structure used to bind parameter and prepared statements.

Syntax

```
typedef struct a_sqlany_bind_param
```

Members

All members of a_sqlany_bind_param, including inherited members.

Variables

Modifier and Type	Variable	Description
public a_sqlany_data_direction	direction	The direction of the data. (input, output, input_output)
public a_sqlany_data_value	value	The actual value of the data.
public char *	name	Name of the bind parameter. This is only used by sqlany_describe_bind_param().

Remarks

To view examples of the a_sqlany_bind_param structure in use, see any of the following sample files in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

Related Information

sqlany_execute(a_sqlany_stmt *) Method [page 18]

1.60 a_sqlany_bind_param_info Structure

Gets information about the currently bound parameters.

Syntax

```
typedef struct a_sqlany_bind_param_info
```

Members

All members of a_sqlany_bind_param_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public char *	name	A pointer to the name of the parameter.
public a_sqlany_data_direction	direction	The direction of the parameter.
public a_sqlany_data_value	input_value	Information about the bound input value.
public a_sqlany_data_value	output_value	Information about the bound output value.
public a_sqlany_native_type	native_type	The native type of the column in the database.
public unsigned short	precision	The precision.
public unsigned short	scale	The scale.
public size_t	max_size	The maximum size a data value in this column can take.

Remarks

sqlany_get_bind_param_info() can be used to populate this structure.

To view examples of the a_sqlany_bind_param_info structure in use, see any of the following sample files in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

Related Information

sqlany_execute(a_sqlany_stmt *) Method [page 18]

1.61 a_sqlany_column_info Structure

Returns column metadata information.

Syntax

```
typedef struct a_sqlany_column_info
```

Members

All members of a_sqlany_column_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public char *	name	The name of the column (null-terminated). The string can be referenced as long as the result set object is not freed.
public a_sqlany_data_type	type	The column data type.
public a_sqlany_native_type	native_type	The native type of the column in the database.
public unsigned short	precision	The precision.
public unsigned short	scale	The scale.
public size_t	max_size	The maximum size a data value in this column can take.
public sacapi_bool	nullable	Indicates whether a value in the column can be null.
public char *	table_name	The name of the table (null-terminated). The string can be referenced as long as the result set object is not freed.
public char *	owner_name	The name of the owner (null-terminated). The string can be referenced as long as the result set object is not freed.
public sacapi_bool	is_bound	Indicates whether the column is bound to a user buffer.
public a_sqlany_data_value	binding	Information about the bound column.

Remarks

sqlany_get_column_info() can be used to populate this structure.

To view an example of the a_sqlany_column_info structure in use, see the following sample file in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

- dbcapi_isql.cpp

1.62 a_sqlany_data_info Structure

Returns metadata information about a column value in a result set.

Syntax

```
typedef struct a_sqlany_data_info
```

Members

All members of a_sqlany_data_info, including inherited members.

Variables

Modifier and Type	Variable	Description
public a_sqlany_data_type	type	The type of the data in the column.
public sacapi_bool	is_null	Indicates whether the last fetched data is NULL. This field is only valid after a successful fetch operation.
public size_t	data_size	The total number of bytes available to be fetched. This field is only valid after a successful fetch operation.

Remarks

sqlany_get_data_info() can be used to populate this structure with information about what was last retrieved by a fetch operation.

To view an example of the a_sqlany_data_info structure in use, see the following sample file in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

Related Information

sqlany_get_data_info(a_sqlany_stmt *, sacapi_u32, a_sqlany_data_info *) Method [page 31]

1.63 a_sqlany_data_value Structure

Returns a description of the attributes of a data value.

Syntax

```
typedef struct a_sqlany_data_value
```

Members

All members of a_sqlany_data_value, including inherited members.

Variables

Modifier and Type	Variable	Description
public char *	buffer	A pointer to user supplied buffer of data.
public size_t	buffer_size	The size of the buffer.
public size_t *	length	A pointer to the number of valid bytes in the buffer. This value must be less than buffer_size.
public a_sqlany_data_type	type	The type of the data.
public sacapi_bool *	is_null	A pointer to indicate whether the last fetched data is NULL.
public sacapi_bool	is_address	Indicates whether the buffer value is an pointer to the actual value.

Remarks

To view examples of the a_sqlany_data_value structure in use, see any of the following sample files in the `sdk\dbcapi\examples` directory of your SQL Anywhere installation.

- dbcapi_isql.cpp
- fetching_a_result_set.cpp
- send_retrieve_full_blob.cpp
- preparing_statements.cpp

1.64 SQLAnywhereInterface Structure

The SQL Anywhere C API interface structure.

Syntax

```
typedef struct SQLAnywhereInterface
```

Members

All members of SQLAnywhereInterface, including inherited members.

Variables

Modifier and Type	Variable	Description
public void *	dll_handle	DLL handle.
public int	initialized	Flag to know if initialized or not.
public void *	sqlany_init	Pointer to sqlany_init() function.
public void *	sqlany_fini	Pointer to sqlany_fini() function.
public void *	sqlany_new_connection	Pointer to sqlany_new_connection() function.
public void *	sqlany_free_connection	Pointer to sqlany_free_connection() function.
public void *	sqlany_make_connection	Pointer to sqlany_make_connection() function.
public void *	sqlany_connect	Pointer to sqlany_connect() function.
public void *	sqlany_disconnect	Pointer to sqlany_disconnect() function.
public void *	sqlany_execute_immediate	Pointer to sqlany_execute_immediate() function.
public void *	sqlany_prepare	Pointer to sqlany_prepare() function.
public void *	sqlany_free_stmt	Pointer to sqlany_free_stmt() function.
public void *	sqlany_num_params	Pointer to sqlany_num_params() function.
public void *	sqlany_describe_bind_param	Pointer to sqlany_describe_bind_param() function.
public void *	sqlany_bind_param	Pointer to sqlany_bind_param() function.
public void *	sqlany_send_param_data	Pointer to sqlany_send_param_data() function.

Modifier and Type	Variable	Description
public void *	sqlany_reset	Pointer to sqlany_reset() function.
public void *	sqlany_get_bind_param_info	Pointer to sqlany_get_bind_param_info() function.
public void *	sqlany_execute	Pointer to sqlany_execute() function.
public void *	sqlany_execute_direct	Pointer to sqlany_execute_direct() function.
public void *	sqlany_fetch_absolute	Pointer to sqlany_fetch_absolute() function.
public void *	sqlany_fetch_next	Pointer to sqlany_fetch_next() function.
public void *	sqlany_get_next_result	Pointer to sqlany_get_next_result() function.
public void *	sqlany_affected_rows	Pointer to sqlany_affected_rows() function.
public void *	sqlany_num_cols	Pointer to sqlany_num_cols() function.
public void *	sqlany_num_rows	Pointer to sqlany_num_rows() function.
public void *	sqlany_get_column	Pointer to sqlany_get_column() function.
public void *	sqlany_get_data	Pointer to sqlany_get_data() function.
public void *	sqlany_get_data_info	Pointer to sqlany_get_data_info() function.
public void *	sqlany_get_column_info	Pointer to sqlany_get_column_info() function.
public void *	sqlany_commit	Pointer to sqlany_commit() function.
public void *	sqlany_rollback	Pointer to sqlany_rollback() function.
public void *	sqlany_client_version	Pointer to sqlany_client_version() function.
public void *	sqlany_error	Pointer to sqlany_error() function.
public void *	sqlany_sqlstate	Pointer to sqlany_sqlstate() function.
public void *	sqlany_clear_error	Pointer to sqlany_clear_error() function.
public void *	sqlany_init_ex	Pointer to sqlany_init_ex() function.
public void *	sqlany_fini_ex	Pointer to sqlany_fini_ex() function.
public void *	sqlany_new_connection_ex	Pointer to sqlany_new_connection_ex() function.
public void *	sqlany_make_connection_ex	Pointer to sqlany_make_connection_ex() function.
public void *	sqlany_client_version_ex	Pointer to sqlany_client_version_ex() function.
public void *	sqlany_cancel	Pointer to sqlany_cancel() function.

Modifier and Type	Variable	Description
public void *	sqlany_register_callback	Pointer to sqlany_register_callback() function.
public void *	sqlany_set_batch_size	Pointer to sqlany_set_batch_size() function.
public void *	sqlany_set_param_bind_type	Pointer to sqlany_set_param_bind_type() function.
public void *	sqlany_get_batch_size	Pointer to sqlany_get_batch_size() function.
public void *	sqlany_set_rowset_size	Pointer to sqlany_set_rowset_size() function.
public void *	sqlany_get_rowset_size	Pointer to sqlany_get_rowset_size() function.
public void *	sqlany_set_column_bind_type	Pointer to sqlany_set_column_bind_type() function.
public void *	sqlany_bind_column	Pointer to sqlany_bind_column() function.
public void *	sqlany_clear_column_bindings	Pointer to sqlany_clear_column_bindings() function.
public void *	sqlany_fetched_rows	Pointer to sqlany_fetched_rows() function.
public void *	sqlany_set_rowset_pos	Pointer to sqlany_set_rowset_pos() function.

Related Information

sqlany_initialize_interface(SQLAnywhereInterface *, const char *) Method [page 36]

1.65 _sacapi_entry_ Variable

The run-time calling convention in use (Windows only).

Syntax

```
#define _sacapi_entry_
```

1.66 SACAPI_ERROR_SIZE Variable

If the command line does not specify which version to build, then build the latest version.

Syntax

```
#define SACAPI_ERROR_SIZE
```

Remarks

Returns the minimal error buffer size.

1.67 SQLANY_API_VERSION_1 Variable

Version 1 was the initial version of the C/C++ API.

Syntax

```
#define SQLANY_API_VERSION_1
```

Remarks

You must define _SACAPI_VERSION as 1 or higher for this functionality.

1.68 SQLANY_API_VERSION_2 Variable

Version 2 introduced the "_ex" functions and the ability to cancel requests.

Syntax

```
#define SQLANY_API_VERSION_2
```

Remarks

You must define _SACAPI_VERSION as 2 or higher for this functionality.

1.69 SQLANY_API_VERSION_3 Variable

Version 3 introduced the "callback" function.

Syntax

```
#define SQLANY_API_VERSION_3
```

Remarks

You must define _SACAPI_VERSION as 3 or higher for this functionality.

1.70 SQLANY_API_VERSION_4 Variable

Version 4 introduced NCHAR support and wide inserts.

Syntax

```
#define SQLANY_API_VERSION_4
```

Remarks

You must define _SACAPI_VERSION as 4 or higher for this functionality.

1.71 SQLANY_CALLBACK Variable

Callback function type.

Syntax

```
#define SQLANY_CALLBACK
```

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP