Oracle® Hospitality RES 3700

Database Access API

Release RES 5.4.1 or higher
E93460-04
June 2019

Oracle Hospitality RES 3700 Database Access API, Release RES 5.4.1 or higher

E93460-04

Contents

Preface

This document provides reference for functions to directly access the RES database.

Audience

This document is intended for developers and programmers with a moderate level of skill and experience with SQL as well as knowledge of the RES database structure.

Important Information

This document is not intended to provide comprehensive training

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received and any associated log files
- Screenshots of each step taken

Documentation

Product documentation is available on the Oracle Help Center at http://docs.oracle.com/en/industries/food-beverage/

Revision History

Date	Description of Change
February 2018	Initial publication.
January 2019	Document overhaul to reflect accurate information.
April 2019	Second document overhaul to reflect accurate information.
June 2019	Column Size Limitation details added.

1

Considerations

The RES Database Access API provides direct access to the RES 3700 embedded database. It is implemented as a set of methods exported in a C-style interface, using __cdecl calling conventions, in a dynamic link library named MdsSysUtilsProxy.dll. This dll contains many functions used broadly throughout core RES applications.

Developers are responsible for any adverse side effects that may occur when designing and implementing software that uses this API. Developers must consider all impacts to the core RES 3700 product, which includes but is not limited to the following:

- **Security** – This API requires the use of database credentials. These sensitive materials MUST be treated with the appropriate measures to prevent any disclosure to a malicious party. It is the merchant's responsibility to ensure that custom applications installed on RES 3700 do not affect the merchant's PCI status.
- **Performance** – Ensure the prevention of locking tables or over-burdening the database server, which may affect RES 3700 performance.
- **Integrity** – Ensure the prevention of altering transaction data generated by RES 3700.
- DO NOT use functions of this dll that are not explicitly documented in this guide.
- DO NOT move, copy, or rename any core RES files.

Installation and Configuration

The use of the RES Database Access API, MdsSysUtils.dll, requires the local computer to be a properly installed and configured Network Node within a RES system. If using this API on a machine that is not a RES server, then the Oracle Hospitality Client Application Loader (CAL) must be used to install the RES application software, and define the computer in POS Configurator | Devices | Network Node.

If using a computer that does not have the CAL software pre-installed, obtain it from the following areas:

- Oracle Software Delivery Cloud https://edelivery.oracle.com
- For RES 5.6 and later, it is installed on the RES Server .\MICROS\Win32CALClientSetup

If using RES 5.5 or higher, the machine must have the correct Client Trust Passphrase configured using the Client Trust Utility, **ctutils.exe**. This is necessary as the API uses the core RES security infrastructure to protect the data involved. Refer to the published installation documentation for the specific version of RES for more details.

The computer where this API is used does not need to be defined as a User Workstation, nor does it need to run POS Operations. Custom applications should not install files into the .\MICROS*\Bin\ folders. These folders are not appropriate for third party files.

Permissions for this and other folders are restricted. Patching clients typically delete the contents of these folders and recreate them with the core files. Custom files may be installed into the .\MICROS*\etc\ folders. It is recommended that the custom applications be installed outside of the .\MICROS\ folder structure.

Patching

The RES patching mechanism is not capable of replacing files that are locked at the time of patching. The patch installation process stops all core RES applications and services in order to enable successful patching. If a custom application is created using the RES Database Access API, or any other core RES files (such as Transaction Services), then it is the responsibility of the creator to ensure their application is stopped before any attempt is made to patch the RES system.

In order to facilitate actions like stopping custom applications, RES patches support custom hooks for executing CustomPre/PostUpdate batch files. Refer to the specific RES version Installation Guide for further details.

Theory of Operation

The RES Database Access API is designed to be used in a single threaded manner. The API stores the parameters in global memory when sqlInitConnection() is called, and then uses those parameters when sqlExecuteQuery() or sqlGetRecordSet() are called. When sqlGetRecordSet() is called, the resulting set is stored locally in global memory. This global memory is created and maintained as 1 cache per process. Due to this design, the user should call all the methods needed in a sequential manner.

Specifically, an application should always call sqlInitConnection() prior to calling sqlExecuteQuery() or sqlGetRecordSet(). If sqlGetRecordSet() is used, then the application should call sqlGetFirst() and then loop call sqlGetNext() until the entire results set is retrieved. If using the API within a multi-threaded application, then the design must ensure that separate threads cannot interrupt this series of calls.

When the API is used from within the System Interface Module (SIM), it is not recommended to call sqlInitConnection() once during an initialization routine. Since other SIM scripts may be in use that are using different database credentials, each SIM script should call sqlInitConnection() every time before calling sqlExecuteQuery() or sqlGetRecordSet().

Version Differences

Certain methods defined in the following chapters have different interfaces, depending on the version of RES installed. The example in Chapter 4 – Sample Project demonstrates how to handle this variation in SIM. If creating a custom application, obtain the RES version from either of the following areas:

1. The environment variable: MICROS_Current_Installation
2. Registry key: HKLM\Software\[WOW6432]\MICROS\

   String Value: Version

Column Size Limitation

In previous RES 3700 versions, the data returned in a single column was limited to 2047 characters. If the data exceeded that limitation, the column would truncate the data at exactly 2047 characters, without notifying the user that the truncation occurred.

Starting with RES v5.7.5, the column size limitation is set to 32,767 bytes, which is the maximum allowable column size supported by the SQL Anywhere database.

The following rules define the behavior of RES 5.7.5 and later:

- If a query attempts to return more than 32,767 bytes, then the column data truncates.
- If the Database Access API cannot allocate adequate memory to a column, then the column data truncates to 2047 characters.
- If truncation occurs, then the column separator - which follows the data - displays as a colon rather than a semicolon.

Refer to szOutRecordData Format for additional formatting details.

2
Database Access Functions

This chapter presents functions that provide access to the RES database. Database access is implemented using the MdsSysUtilsProxy DLL. This DLL is only available in RES v5.4.1 and later. All functions use __cdecl calling conventions.

The following functions are available through this DLL.

- Connect to Database (sqlInitConnection)
- Close Database Connection (sqlCloseConnection)
- Check Whether the Database Connection is Open (sqlIsConnectionOpen)
- Execute SQL Statement (sqlExecuteQuery)
- Get the Result Set (sqlGetRecordSet)
- Get the First Record in the Result Set (sqlGetFirst)
- Get the Next Record in the Result Set (sqlGetNext)
- Get the Last Record in the Result Set (sqlGetLast)
- Get the Previous Record in the Result Set (sqlGetPrev)
- Get the Last Error (sqlGetLastErrorString)

An application typically invokes the functions in the following sequence:

1. Loads the MdsSysUtilsProxy DLL.
2. Calls `sqlInitConnection` to establish the connection parameters to the database.
3. Calls `sqlExecuteQuery` to execute that has no result set.

   OR
4. Calls one of the following:
   a. `sqlGetRecordSet` to execute a query and retrieve the result set to a local cache.
   b. `sqlGetFirst`, `sqlGetNext`, `sqlGetLast`, and `sqlGetPrev` functions to navigate and read a row at a time from the locally cached result set.
5. Calls the `sqlGetLastErrorString` after certain calls to test for success and access error information.
6. Calls `sqlCloseConnection` to close the connection to the database.

Connect to Database

```
BOOL sqlInitConnection( const char *szDbName,
                        const char *szConnInfo,
                        const char *unused );
```

Typically, this is the first function called. It establishes connection parameters that will be used when a query is executed. No communication occurs with the DB server at this time.

Parameters

Parameter	Description
szDbName	Database name; in rare circumstances it is known as “MICROS”. Input parameter. Memory is managed by the calling function.
szConnInfo	Connection string, typically 256 char wide. Using the following format: “`ODBC;UID=`*`username`*`;PWD=`*`password`*”. Input parameter. Memory is managed by the calling function.
unused	Recommend passing “ “ (a string with 1 space). In version 5.7.3 and earlier, passing 0 or “” will result in a warning within the 3700d.log.

Return Value

Returns 1 if valid parameters pass and the system’s encryption environment is properly prepared.

Returns 0 if an invalid parameter is passed, or if the application is unable to use the underlying encryption capabilities of RES.

Close Database Connection

```
BOOL sqlCloseConnection();
```

Typically, this is the last function to be called. It clears the parameters stored by any prior call to sqlInitConnection. Releases any memory used to cache the results from a prior call sqlGetRecordSet.

Parameters

None

Return Value

Returns 1, always.

Check whether the Database Connection is Open

```
BOOL sqlIsConnectionOpen( int *isOpen );
```

This function indicates if a prior call to sqlInitConnection has succeeded.

Parameters

Parameter	Description
isOpen	Pointer to an integer. Output parameter is set to 1 if the most recent call to sqlInitConnection succeeds; otherwise, set to 0.

Return Value

Returns 1 if the most recent call to sqlInitConnection succeeds.

Returns 0 if the most recent call to sqlInitConnection fails.

Execute SQL Statement

```
BOOL sqlExecuteQuery( const char *szCmd );
```

Executes the sql query against the database, using the connection parameters previously passed with sqlInitConnection. This function does not support retrieving the result set of the query. It can be used to call stored procedures or other queries where a result set is not needed.

Parameters

Parameter	Description
szCmd	SQL query, such as the string: `“UPDATE MICROS.gss_restaurant_def SET enabled = ‘T’”` Input parameter. Memory is managed by the calling function.

Return Value

Returns 1 if the query execution is successful.

Returns 0 if there is an error.

Calling sqlGetLastErrorString retrieves error information.

Execute Query and Get the Result Set

```
BOOL sqlGetRecordSet( const char *szCmd );
```

Executes the sql query against the database, using the connection parameters previously passed with sqlInitConnection. If the connection and query are successful, then the result set returns and is cached locally within memory managed by this call.

Parameters

Parameter	Description
szCmd	SQL query, such as the string: `"SELECT * from MICROS.gss_restaurant_def"` Input parameter. Memory is managed by the calling function.

Return Value

Returns 1 if the connection, query, and retrieval are all successful.

Returns 0 if there is an error.

Calling sqlGetLastErrorString retrieves error information.

Get the First Record in the Result Set

```
void sqlGetFirst( char *szOutRecordData );//PRE-5.6

void sqlGetFirst( char *szOutRecordData,
                  const size_t dataSize );
```

Initializes the cached record set at the beginning and retrieves the first record/row from the result set.

Parameters

Parameter	Description
szOutRecordData	Pointer to the memory where the first record of the result set will be copied. The caller is responsible for allocating and managing the memory used by this parameter. On return, this parameter contains null terminated data.
dataSize	Size (in bytes) of the memory allocated by the caller for szOutRecordData. This parameter was introduced in RES v5.6. It is critical that the application detects the version of RES in use and calls the function with the correct number of parameters based on the specific version of RES.

Return Value

None

Get the Next Record in the Result Set

```
void sqlGetNext( char *szOutRecordData );//PRE-5.6

void sqlGetNext( char *szOutRecordData,
                 const size_t dataSize );
```

Gets the next record from the record set.

Parameters

Parameter	Description
szOutRecordData	Pointer to the memory where the first record of the result set will be copied. The caller is responsible for allocating and managing the memory used by this parameter. On return, this parameter contains null terminated data.
dataSize	Size (in bytes) of the memory allocated by the caller for szOutRecordData. This parameter was introduced in RES v5.6. It is critical that the application detects the version of RES in use and calls the function with the correct number of parameters based on the specific version of RES.

Return Value

None

Get the Last Record in the Result Set

```
void sqlGetLast( char *szOutRecordData );//PRE-5.6

void sqlGetLast( char *szOutRecordData,
                 const size_t dataSize );
```

Gets the last record from the record set.

Parameters

Parameter	Description
szOutRecordData	Pointer to the memory where the first record of the result set will be copied. The caller is responsible for allocating and managing the memory used by this parameter. On return, this parameter contains null terminated data.
dataSize	Size (in bytes) of the memory allocated by the caller for szOutRecordData. The parameter is not altered by the function.

Parameter	Description
	This parameter was introduced in RES v5.6. It is critical that the application detects the version of RES in use and calls the function with the correct number of parameters based on the specific version of RES.

Return Value

None

Get the Previous Record in the Result Set

```
void sqlGetPrev( char *szOutRecordData );//PRE 5.6

void sqlGetPrev( char *szOutRecordData,
                 const size_t dataSize );
```

Gets the previous record from the record set.

Parameters

Parameter	Description
szOutRecordData	Pointer to the memory where the first record of the result set will be copied. The caller is responsible for allocating and managing the memory used by this parameter. On return, this parameter contains null terminated data.
dataSize	Size (in bytes) of the memory allocated by the caller for szOutRecordData. The parameter is not altered by the function. This parameter was introduced in RES v5.6. It is critical that the application detects the version of RES in use and calls the function with the correct number of parameters based on the specific version of RES.

Return Value

None.

Get the Last Error

```
void sqlGetLastErrorString( char *szError );//PRE5-6

void sqlGetLastErrorString( char *szError,
                            const size_t dataSize );
```

Retrieves a string representing the last error that occurred in the context of the previous call.

Parameters

Parameter	Description
szError	Pointer to the memory where an error message may be copied. The caller is responsible for allocating and managing the memory used by this parameter. On return, this parameter contains null terminated data.
dataSize	Size (in bytes) of the memory allocated by the caller for szError. The parameter is not altered by the function. This parameter was introduced in RES v5.6. It is critical that the application detects the version of RES in use and calls the function with the correct number of parameters based on the specific version of RES.

Return Value

None

3
szOutRecordData Format

Each record of the result set, as returned by calls to sqlGetFirst/Next/Prev/Last(), contains data in the following format:

Field 1;Field 2;…;Field N;

The data is delimited with semicolons. If the column data truncates, then a colon is used as the delimiter for that column. The escape character is **/** (forward slash). Therefore, any instance of ":", ";", or "/" in the data, will be represented as "/:", "/;", or "//", respectively.

For example, a series of 3 fields containing "ab:c", "ab;c", and "ab/c" would be presented as:

ab/:c;ab/;c;ab//c;

> **NOTE:**
>
> The use of "/" as the escape character is specific to RES v5.7.3 and later.
>
> The use of ":" to indicate column truncation is specific to RES v5.7.5 and later.

Null Data

Fields with NULL contain no data. These fields are delimited.

For example, a series of 5 fields containing "null, abc, null, null, def", would be presented as:

;abc;;;def;

Binary Data

Fields containing binary data may need to be converted using the SQL command BINTOHEX(). This avoids issues of attempting to pass zeros in string data types.

4 Sample Project

The System Interface Module (SIM) and Integrated Scripting Language (ISL) are interchangeable terms that refer to a proprietary scripting language supported by RES and other Oracle Hospitality POS Products.

The chapter contains a sample SIM code to test for open checks at Service total, and to display information about them.

It demonstrates a way to use a proper number of parameters for several functions based on a specific RES version.

SIM Project Code

```
// Check For Open Checks and get open check information
// A script to demonstrate the usage of RES SIM DB API

// SIM OPTIONS
RetainGlobalVar

// Global variables
var _MDSSYSUTILSPROXY_HANDLE: N20
var NumOpenChecks: N3
var _API_NEW : N1

sub CheckAPIVersion()
    var maj : N9
    var min : N9
    var rel : N9
    var bld : N9

    _API_NEW = 0
    infomessage "test"
    split @VERSION, ".", maj, min, rel, bld
    if (( maj >= 5 ) and (min >= 6 )) or maj > 5
        _API_NEW = 1
    endif
endsub

sub MDSUtils_Init()
    if (_MDSSYSUTILSPROXY_HANDLE = 0)
        dllLoad _MDSSYSUTILSPROXY_HANDLE, "MDSSysUtilsProxy.dll"
        call CheckAPIVersion()
    endif
    if (_MDSSYSUTILSPROXY_HANDLE = 0)
        exitWithError "Failed to load MDSSysUtilsProxy.dll"
    endif
endsub

sub MDSUtils_Deinit()
    if (_MDSSYSUTILSPROXY_HANDLE)
        dllfree _MDSSYSUTILSPROXY_HANDLE
        _MDSSYSUTILSPROXY_HANDLE = 0
    endif
endsub

sub dbOpen()
    var username     : A32
    var password     : A32
    var connString: A256

    username = "user"    // provide db user
    password = "pass"    // provide db password

    call MDSUtils_Init()
```

```
    if (_MDSSYSUTILSPROXY_HANDLE)
        format connString as "ODBC;UID=", username, ";PWD=", password
        dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlInitConnection("micros", connString, " ")
    endif
endsub

sub dbClose()
    if (_MDSSYSUTILSPROXY_HANDLE)
        dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlCloseConnection()
    endif
endsub

sub dbExecute(              \
        var dbHandle: N10,  \
        ref sqlStmt,        \
        ref sqlErr          \
    )
    sqlErr = ""
    if (dbHandle)
        dllCall_cdecl dbHandle, sqlExecuteQuery(ref sqlStmt)
        if (_API_NEW = 1)
            dllCall_cdecl dbHandle, sqlGetLastErrorString(ref sqlErr, varsize(sqlErr))
        else
            dllCall_cdecl dbHandle, sqlGetLastErrorString(ref sqlErr)
        endif

        if (sqlErr <> "")
            call ODBCError()
        endif
    else
        sqlErr = "Invalid database handle"
    endif
endsub

sub dbGetSingleRowResult(   \
        ref sqlStmt,        \
        ref sqlRslt,        \
        ref sqlErr          \
    )
    sqlErr = ""
    sqlRslt = ""
    if (_MDSSYSUTILSPROXY_HANDLE)
        dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetRecordSet(ref sqlStmt)
        if (_API_NEW = 1)
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetLastErrorString(ref sqlErr,
varsize(sqlErr))
        else
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetLastErrorString(ref sqlErr)
        endif
        if (sqlErr = "")
            call dbGetFirst(sqlRslt, sqlErr)
        else
            call ODBCError()
        endif
    else
        sqlErr = "Invalid database handle"
    endif
endsub

sub dbGetRecordSet(         \
        ref sqlStmt,        \
        ref sqlRslt,        \
        ref sqlErr          \
    )
    sqlErr = ""
    sqlRslt = ""
    if (_MDSSYSUTILSPROXY_HANDLE)
        call dbGetSingleRowResult(sqlStmt, sqlRslt, sqlErr)
    else
        sqlErr = "Invalid database handle"
    endif
endsub

sub dbGetFirst(             \
        ref sqlRslt,        \
        ref sqlErr          \
    )
    sqlRslt = ""
    if (_MDSSYSUTILSPROXY_HANDLE)
        if (_API_NEW = 1)
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetFirst(ref sqlRslt, varsize(sqlRslt))
```

```
        else
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetFirst(ref sqlRslt)
        endif
    else
        sqlErr = "Invalid database handle"
    endif
endsub
sub dbGetNext(              \
        ref sqlRslt,        \
        ref sqlErr          \
    )
    sqlRslt = ""
    sqlErr = ""

    if (_MDSSYSUTILSPROXY_HANDLE)
        if (_API_NEW = 1)
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetNext(ref sqlRslt, varsize(sqlRslt))
        else
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetNext(ref sqlRslt)
        endif
    else
        sqlErr = "Invalid database handle"
    endif
endsub
sub dbGetLast(              \
        ref sqlRslt,        \
        ref sqlErr          \
    )
    sqlRslt = ""
    if (_MDSSYSUTILSPROXY_HANDLE)
        if (_API_NEW = 1)
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetLast(ref sqlRslt, varsize(sqlRslt))
        else
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetLast(ref sqlRslt)
        endif
    else
        sqlErr = "Invalid database handle"
    endif
endsub
sub dbGetPrevious(          \
        ref sqlRslt,        \
        ref sqlErr          \
    )
    sqlRslt = ""
    if (_MDSSYSUTILSPROXY_HANDLE)
        if (_API_NEW = 1)
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetPrevious(ref sqlRslt,
varsize(sqlRslt))
        else
            dllCall_cdecl _MDSSYSUTILSPROXY_HANDLE, sqlGetPrevious(ref sqlRslt)
        endif
    else
        sqlErr = "Invalid database handle"
    endif
endsub

sub ODBCError()
    call MDSUtils_Deinit()
endsub

sub getOpenChecks()
    var sqlStmt         : A500
    var sqlRslt         : A2000
    var sqlErr          : A255

    NumOpenChecks = 0

    format sqlStmt as "select count() from micros.chk_dtl where chk_open = 'T' ", \
                      "and chk_clsd_date_time is null and
DATEADD(minute,30,chk_open_date_time) < NOW(*) ", \
                      "and rvc_seq = (select rvc_seq from micros.rvc_def where obj_num = ",
@RVC, ");"

    call dbOpen()

    call dbGetRecordSet(sqlStmt, sqlRslt, sqlErr)
    if (sqlErr <> "")
        call dbClose()
        errormessage sqlErr
        exitCancel
    endif
```

```
    if (sqlRslt <> "")
        NumOpenChecks = sqlRslt
    endif

    call dbClose()
endsub

sub getCheckInfo()
    var sqlStmt         : A500
    var sqlRslt         : A2000
    var sqlErr          : A255
    var chkNum            : A10
    var chkDateTime     : A30

    format sqlStmt as "select chk_num, chk_open_date_time from micros.chk_dtl where chk_open =
'T' ", \
                      "and chk_clsd_date_time is null and
DATEADD(minute,30,chk_open_date_time) < NOW(*) ", \
                      "and rvc_seq = (select rvc_seq from micros.rvc_def where obj_num = ",
@RVC, ");"

    call dbOpen()
    call dbGetRecordSet(sqlStmt, sqlRslt, sqlErr)
    if (sqlErr <> "")
        call dbClose()
        errormessage sqlErr
        exitCancel
    endif

    while (sqlRslt <> "")
        split sqlRslt, ";", chkNum, chkDateTime

        Infomessage "Open Check information", chkNum, " ", chkDateTime

        call dbGetNext(sqlRslt, sqlErr)
    endwhile

    call dbClose()
endsub

//
// Event called on tender/media service total type
//
event srvc_total: XXX    // replace XXX with tender/media object number
    var message : A500

    if (@inbackupmode = 0 and @instandalonemode = 0)    // Check for a connection with the
database server
        call getOpenChecks()

        if NumOpenChecks > 0
            format message as NumOpenChecks, " Check(s) open for longer than 30 minutes"
            Infomessage message
        endif

        call getCheckInfo()
    endif
endevent

//
// Event called when POS client exits.
// DB connection is closed and client DLL is unloaded.
//
event exit
    call dbClose()
    call MDSUtils_Deinit()
endevent
```