Programming Extended Stored Procedures

Microsoft Open Data Services provides a server-based application programming interface (API) for extending Microsoft® SQL ServerTM functionality. The API consists of C and C++ functions and macros used to build applications in the following categories:

- Extended stored procedures.
- Gateway applications.

Extended Stored Procedures

Packaged as dynamic-link libraries (DLLs), extended stored procedures provide a way to extend SQL Server functionality through functions developed by using C/C++, the Open Data Services API, and the Microsoft Win32® API. These functions can send result sets and output parameters back to the client from a variety of external data sources.

Gateway Applications

In the past, Open Data Services was used to write server applications, such as gateways to non-SQL Server database environments. With the emergence of newer and more powerful technologies, such as Windows NT Component Services and SQL Server distributed queries, the need for Open Data Services gateway applications has largely been replaced.

- If you have existing gateway applications, you cannot use opends60.dll and ums.dll that are shipped with SQL Server 2000 to run the applications. Gateway applications are no longer supported.
- You can continue to run gateway applications using the opends60.dll and ums.dll that shipped with SQL Server 7.0.

How Extended Stored Procedures Work

The process by which an extended stored procedure works is:

- 1. When a client executes an extended stored procedure, the request is transmitted in tabular data stream (TDS) format from the client application through the Net-Libraries and Open Data Services to Microsoft® SQL Server[™].
- 2. SQL Server searches for the DLL associated with the extended stored procedure, and loads the DLL if it is not already loaded.
- 3. SQL Server calls the requested extended stored procedure (implemented as a function inside the DLL).
- 4. The extended stored procedure passes result sets and return parameters back to the server by using the Open Data Services API.

Execution Characteristics of Extended Stored Procedures

The execution of an extended stored procedure has these characteristics:

- The extended stored procedure function is executed under the security context of Microsoft® SQL Server[™].
- The extended stored procedure function runs in the process space of SQL Server.
- The Microsoft Win32® thread associated with the execution of the extended stored procedure is the same one used for the client connection.

After the extended stored procedure DLL is loaded, the DLL remains loaded in the address space of the server until the SQL Server is stopped or the administrator explicitly unloads the DLL by using DBCC *DLL_name* (FREE). This is the default behavior.

The extended stored procedure can be executed from Transact-SQL as a stored procedure by using the EXECUTE statement:

EXECUTE @retval = xp_extendedProcName @param1, @param2 OUTPUT

where

@retval

Is a return value.

@param1

Is an input parameter.

@param2

Is an input/output parameter.

CAUTION Extended stored procedures offer performance enhancements and

extend SQL Server functionality. However, because the extended stored procedure DLL and SQL Server share the same address space, a problem procedure can adversely affect SQL Server functioning. Although exceptions thrown by the extended stored procedure DLL are handled by SQL Server, it is possible to damage SQL Server data areas. As a security precaution, only SQL Server system administrators can add extended stored procedures to SQL Server. These procedures should be thoroughly tested before they are installed.

Creating Extended Stored Procedures

An extended stored procedure is a function with a prototype:

SRVRETCODE xp_extendedProcName (SRVPROC *);

Using the prefix "xp_" is optional. Extended stored procedure names are case sensitive when referenced in Transact-SQL statements, regardless of code page/sort order installed on the server. An extended stored procedure is implemented in a 32-bit dynamic-linked library (DLL). When you build a DLL:

• If an entry point is necessary, write a **DllMain** function.

This function is optional; if you do not provide it in source code, the compiler links its own version, which does nothing but return TRUE. If you provide a **DllMain** function, the operating system calls this function when a thread or process attaches to or detaches from the DLL.

• All functions called from outside the DLL (all extended stored procedure functions) must be exported.

You can export a function by listing its name in the EXPORTS section of a .def file, or you can prefix the function name in the source code with __declspec(dllexport), a Microsoft compiler extension (Note that __declspec() begins with two underscores).

These Open Data Services files are required for creating an extended stored procedure DLL.

File	Description
Srv.h	Open Data Services header file
Opends60.lib	Import library for Opends60.dll

It is highly recommended that all Microsoft® SQL Server[™] 2000 extended stored procedure DLLs implement and export the following function:

```
__declspec(dllexport) ULONG __GetXpVersion()
```

```
return ODS_VERSION;
```

}

When SQL Server loads an extended stored procedure DLL, SQL Server checks for the above function.

Note ___declspec(dllexport) is a Microsoft-specific compiler extension. If your compiler does not support this directive, you should export this function in your DEF file under the EXPORTS section.

When SQL Server is started with the trace flag -T260 or if a user with system administrator privileges runs DBCC TRACEON (260), then if the extended stored procedure DLL does not support __GetXpVersion(), a warning message (Error 8131: Extended stored procedure DLL '%' does not export __GetXpVersion().) is printed to the error log (Note that __GetXpVersion() begins with two underscores). If you get this message, and you are running an extended stored procedure DLL compiled with headers and libraries from SQL Server version 6.*x*, refer to Level 1: Handling Discontinued Functionality. If you get this message and are running an extended stored procedure DLL compiled with headers and libraries from SQL Server 7.0, your extended stored procedure DLL is not exporting the function __GetXpVersion().

If the extended stored procedure DLL exports __GetXpVersion(), but the version returned by the function is less than that required by the server, a warning message (Error 8132: Extended stored procedure DLL '%' reports its version is %d.%d. Server expects version %d.%d.) stating the version returned by the function and the version expected by the server is printed to the error log. If you get this message, you are returning an incorrect value from __GetXpVersion(), or you are compiling with an older version of srv.h.

Note SetErrorMode, a Microsoft Win32[®] function, should not be called in extended stored procedures.

For more information about creating a DLL, see the development environment documentation and the Microsoft Win32 SDK documentation.

To create an extended stored procedure DLL by using Microsoft Visual C++

- 1. Create a new project of type Win32 Dynamic Link Library.
- 2. Set the directory for **include files** and **library files** to C:\Program

Files\Microsoft SQL Server\80\Tools\DevTools\Include and C:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Lib, respectively.

- a. On the **Tools** menu, click **Options**.
- b. In the **Options** dialog box, click the **Directories** tab and set the directory for **include files** and **library files**.
- 3. On the **Project** menu, click **Settings**.
- 4. In the **Project Settings** dialog box, click the **Link** tab. Click the **General** category, and then add **opends60.lib** to **object/library modules**.
- 5. Add source files (.c, .cpp, and .rc files, and so on) to your project.
- 6. Compile and link your project.

Extended Stored Procedure Sample: xp_hello

These code portions from **xp_hello** illustrate the basics of writing an extended stored procedure. The complete code for this example is available in the Samples\ODS directory. This is in a sample only available if you select Dev Tools in setup.

Transact-SQL Script

```
-- TSQL script exercising xp_hello sample
use master
go
```

```
sp_addextendedproc 'xp_hello', 'xp_hello.dll'
go
```

```
-- Call xp_hello with literal parameters
declare @txt varchar(33)
exec xp_hello @txt OUTPUT
select @txt AS OUTPUT_Parameter
go
```

```
sp_dropextendedproc 'xp_hello'
go
```

```
dbcc xp_hello(free)
go
```

- -- EXPECTED RESULTS
- -- Column 1:

-- -----

-- Hello World!

-- (1 row(s) affected)

-- OUTPUT_PARAMETER

-- Hello World!

-- (1 row(s) affected)

C Source Code

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <windows.h>
#include <srv.h>
#include <time.h>

// Macros -- return codes#define XP_NOERROR 0#define XP_ERROR 1

```
#define MAX_SERVER_ERROR 20000
#define XP_HELLO_ERROR MAX_SERVER_ERROR+1
```

void printUsage (SRV_PROC* pSrvProc); void printError (SRV_PROC *pSrvProc, CHAR* szErrorMsg);

// It is highly recommended that all Microsoft® SQL Server (7.0
// and greater) extended stored procedure DLLs implement and export
// __GetXpVersion. For more information see SQL Server
// Books Online
ULONG __GetXpVersion()

```
{
    return ODS_VERSION;
}
SRVRETCODE xp_hello(SRV_PROC* pSrvProc)
{
    char szText[15] = "Hello World!";
    BYTE bType;
    long cbMaxLen;
    long cbActualLen;
    BOOL fNull;
```

```
#ifdef _DEBUG
```

```
// In a debug build, look up the data type name for assistance.
DBCHAR* pdbcDataType;
int cbDataType;
```

#endif

```
// Count up the number of input parameters. There should only be o
if (srv_rpcparams(pSrvProc) != 1)
    {
      // Send error message and return
      //
      printUsage (pSrvProc);
      return (XP_ERROR);
    }
// Use srv_paraminfo to get data type and length information.
if (srv_paraminfo(pSrvProc, 1, &bType, &cbMaxLen, &cbActualLe
      NULL, &fNull) == FAIL)
    {
}
```

```
printError (pSrvProc, "srv_paraminfo failed...");
```

```
return (XP_ERROR);
    }
  // Make sure first parameter is a return (OUTPUT) parameter
  if ((srv_paramstatus(pSrvProc, 1) & SRV_PARAMRETURN) == FA
    ł
    printUsage (pSrvProc);
    return (XP_ERROR);
    }
  // Make sure first parameter is of char or varchar datatype
  if (bType != SRVBIGVARCHAR && bType != SRVBIGCHAR)
    ł
    printUsage (pSrvProc);
    return (XP_ERROR);
    }
  // Make sure first paramter is large enough to hold data
  if (cbMaxLen < (long)strlen(szText))
    {
    printError (pSrvProc, "output param max. length should be bigger
    return (XP_ERROR);
    }
  // Describe the results set
//#define METHOD1
#ifdef METHOD1
  srv_describe(pSrvProc, 1, "Column 1", SRV_NULLTERM, bType,
      cbMaxLen, bType, strlen(szText), szText);
#else
  srv describe(pSrvProc, 1, "Column 1", SRV NULLTERM, bType,
      cbMaxLen, bType, 0, NULL);
```

```
// Set the column's length
if (srv_setcollen(pSrvProc, 1, strlen(szText)) == FAIL)
{
    printError (pSrvProc, "srv_setcollen failed...");
    return (XP_ERROR);
}
// Set the column's data
if (srv_setcoldata(pSrvProc, 1, szText) == FAIL)
{
    printError (pSrvProc, "srv_setcoldata failed...");
    return (XP_ERROR);
}
```

```
#endif //METHOD1
```

```
#ifdef _DEBUG
```

// A debugging aid. Get the name of the data type of the parameter.
pdbcDataType = srv_symbol(SRV_DATATYPE, (int) bType, &cbDataType

```
// Send a row to client
if (srv_sendrow(pSrvProc) == FAIL)
{
    printError (pSrvProc, "srv_sendrow failed...");
    return (XP_ERROR);
    }
```

```
// Set the output parameter
```

if (FAIL == srv_paramsetoutput(pSrvProc, 1, szText, strlen(szText),
 {
 printError (pSrvProc, "srv_paramsetoutput failed...");
 return (XP_ERROR);

}

```
srv_senddone(pSrvProc, (SRV_DONE_COUNT | SRV_DONE_MC
  return (XP_NOERROR);
}
// send XP usage info to client
void printUsage (SRV_PROC *pSrvProc)
{
 // usage: exec xp_hello <@param1 output>
 // Example:
  // declare @txt varchar(33)
  // exec xp_hello @txt OUTPUT
  // select @txt
  srv_sendmsg(pSrvProc, SRV_MSG_ERROR, XP_HELLO_ERROF
      NULL, 0, (DBUSMALLINT) LINE ,
      "Usage: exec xp_hello <@param1 output>",
      SRV NULLTERM);
  srv_senddone(pSrvProc, (SRV_DONE_ERROR | SRV_DONE_MO
}
// send szErrorMsg to client
void printError (SRV_PROC *pSrvProc, CHAR* szErrorMsg)
{
  srv_sendmsg(pSrvProc, SRV_MSG_ERROR, XP_HELLO_ERROF
      NULL, 0, (DBUSMALLINT) __LINE__,
      szErrorMsg,
      SRV_NULLTERM);
  srv_senddone(pSrvProc, (SRV_DONE_ERROR | SRV_DONE_MO
}
```

Xp_hello.def

The .def file used in the **xp_hello** sample exports the **xp_hello** function. This is in a sample only available if you select Dev Tools during setup.

LIBRARY XP_HELLO

DESCRIPTION 'Sample SQL Server Extended Stored Procedure DL'

EXPORTS xp_hello __GetXpVersion

Debugging an Extended Stored Procedure

Debugging an extended stored procedure is similar to debugging a DLL. For general information about debugging DLLs, see the development environment and debugger vendor documentation.

To debug an extended stored procedure DLL by using Microsoft Visual C++

- 1. In Microsoft® Windows NT® 4.0, stop the SQL Server Service by using SQL Server Service Manager or Control Panel.
- 2. Open the project that builds the extended stored procedure DLL, and build it.
- 3. Copy the extended stored procedure DLL to the SQL Server \Binn directory (C:\Program Files\Microsoft SQL Server\Mssql\Binn by default).
- 4. On the **Project** menu, click **Settings**.
- 5. In the **Project Settings** dialog box, click the **Debug** tab.
- 6. In the **Category** list, click **General**.
- In the Executable for debug session box, enter the path and file name of the Microsoft SQL Server[™] executable file (for example, C:\Program Files\Microsoft SQL Server\Mssql\Binn\Sqlservr.exe). For information about sqlservr arguments, see sqlservr Application.
- 8. In the **Category** list, click **Additional DLLs**.
- 9. In the **Local module name** box, enter the names of any additional

DLLs you want to debug.

- 10. Click **OK** to store the information in the project.
- 11. Set breakpoints as required in the DLL source files or on function symbols in the DLL.
- 12. On the **Build** menu, click **Go** to start the debugger.

See Also

sqlservr Application

Adding an Extended Stored Procedure to SQL Server

A DLL that contains extended stored procedure functions acts as an extension to Microsoft® SQL Server[™]. To install the DLL, copy the file to the directory containing the standard SQL Server DLL files (C:\Program Files\Microsoft SQL Server\Mssql\Binn by default).

To add each extended stored procedure function in an extended stored procedure DLL, a SQL Server system administrator must run the **sp_addextendedproc** system stored procedure, specifying the name of the function and the name of the DLL in which that function resides. For example, this command registers the function **xp_hello**, located in xp_hello.dll, as a SQL Server extended stored procedure:

sp_addextendedproc 'xp_hello', 'xp_hello.dll'

It is not necessary to stop and restart SQL Server. This is in a sample only available if you select Dev Tools during setup.

See Also

sp_addextendedproc

Removing an Extended Stored Procedure from SQL Server

To drop each extended stored procedure function in an extended stored procedure DLL, a Microsoft® SQL ServerTM system administrator must run the **sp_dropextendedproc** system stored procedure, specifying the name of the function and the name of the DLL in which that function resides. For example, this command removes the function **xp_hello**, located in Xp_hello.dll, from SQL Server:

sp_dropextendedproc 'xp_hello'

This is in a sample only available if you select Dev Tools during setup.

See Also

sp_dropextendedproc

Querying Extended Stored Procedures Installed in SQL Server

A Microsoft[®] SQL Server[™] system administrator can display the currently defined extended stored procedures and the name of the DLL to which each belongs by running the **sp_helpextendedproc** system procedure. For example, this command returns the DLL to which **xp_hello** belongs:

sp_helpextendedproc 'xp_hello'

This is in a sample only available if you select Dev Tools during setup.

If **sp_helpextendedproc** is executed without specifying an extended stored procedure, then all the extended stored procedures and their DLLs are displayed.

See Also

sp_helpextendedproc

Unloading an Extended Stored Procedure DLL

Microsoft® SQL Server[™] loads an extended stored procedure DLL as soon as a call is made to one of the functions of the DLL. The DLL remains loaded until the server is shut down or until the system administrator uses the DBCC statement to unload it. For example, this command unloads Xp_hello.dll, allowing the system administrator to copy a newer version of this file to the directory without shutting down the server:

DBCC xp_hello(FREE)

This is in a sample only available if you select Dev Tools during setup.

See Also

DBCC dllname (FREE)

Unicode Data and Server Code Pages

Open Data Services APIs are enabled for Unicode data; however, they are not enabled for Unicode meta data. The #define Unicode directive does not have any effect on the Open Data Services API.

All meta data returned by, or provided to Open Data Services by your Open Data Services application is assumed to be in the multibyte code page of the server. The default code page of an Open Data Services server application is the ANSI code page of the computer on which the application is running, which can be obtained by calling **srv_pfield** with the field parameter set to SRV_SPROC_CODEPAGE.

If your Open Data Services application is Unicode-enabled, you must convert your Unicode meta data column names, error messages, and so on to multibyte data before passing this data to the Open Data Services API.

Example

The following extended stored procedure provides an example of the Unicode conversions discussed. Note that:

- Column data is passed as Unicode data to **srv_describe** because the column is described to be SRVNVARCHAR.
- Column name meta data is passed to **srv_describe** as multibyte data.

The extended stored procedure calls **srv_pfield** with the field parameter set to SRV_SPROC_CODEPAGE to obtain the multibyte code page of Microsoft® SQL Server[™].

• Error messages are passed to **srv_sendmsg** as multibyte data.

```
__declspec(dllexport) RETCODE proc1 (SRV_PROC *srvproc)
{
    #define MAX_COL_NAME_LEN 25
    #define MAX_COL_DATA_LEN 50
```

#define MAX_ERR_MSG_LEN 250 #define MAX_SERVER_ERROR 20000 #define XP_ERROR_NUMBER MAX_SERVER_ERROR+1

int retval; UINT serverCodePage; CHAR *szServerCodePage;

WCHAR unicodeColumnName[MAX_COL_NAME_LEN]; CHAR multibyteColumnName[MAX_COL_NAME_LEN];

WCHAR unicodeColumnData[MAX_COL_DATA_LEN];

WCHAR unicodeErrorMessage[MAX_ERR_MSG_LEN]; CHAR multibyteErrorMessage[MAX_ERR_MSG_LEN];

lstrcpyW (unicodeColumnName, L"column1"); lstrcpyW (unicodeColumnData, L"column1 data"); lstrcpyW (unicodeErrorMessage, L"No Error!");

// Obtain server code page.

//

```
szServerCodePage = srv_pfield (srvproc, SRV_SPROC_CODEPAG
if (NULL != szServerCodePage)
```

```
serverCodePage = atol(szServerCodePage);
```

else

{ // Problem situation exists.

srv_senddone(srvproc, (SRV_DONE_ERROR | SRV_DONE_MC
return 1;

}

// Convert column name for Unicode to multibyte using the // server code page.

```
//
retval = WideCharToMultiByte(
  serverCodePage,
                            // code page
                     // default
  0,
  unicodeColumnName,
                               // wide-character string
  -1,
                     // string is null terminated
  multibyteColumnName, // address of buffer for new
                     // string
  sizeof (multibyteColumnName), // size of buffer
  NULL, NULL);
if (0 == retval)
{
  lstrcpyW (unicodeErrorMessage, L"Conversion to multibyte
  failed.");
  goto Error;
}
retval = srv_describe (srvproc, 1, multibyteColumnName,
SRV NULLTERM,
 SRVNVARCHAR, MAX COL DATA LEN*sizeof(WCHAR), //
  SRVNVARCHAR, lstrlenW(unicodeColumnData)*sizeof(WCHA
  unicodeColumnData); //source
if (FAIL == retval)
{
  lstrcpyW (unicodeErrorMessage, L"srv_describe failed.");
  goto Error;
}
retval = srv_sendrow(srvproc);
if (FAIL == retval)
{
```

```
lstrcpyW (unicodeErrorMessage, L"srv_sendrow failed.");
```

```
goto Error;
  }
  retval = srv_senddone (srvproc, SRV_DONE_MORE|SRV_DONE_
  if (FAIL == retval)
  {
    lstrcpyW (unicodeErrorMessage, L"srv senddone failed.");
    goto Error;
  }
  return 0;
Error:
  // convert error message from Unicode to multibyte.
  retval = WideCharToMultiByte(
    serverCodePage,
                             // code page
                       // default
    0,
    unicodeErrorMessage,
                                // wide-character string
                       // string is null terminated
    -1.
                                // address of buffer for new
    multibyteErrorMessage,
                       // string
    sizeof (multibyteErrorMessage), // size of buffer
    NULL, NULL);
srv_sendmsg(srvproc, SRV_MSG_ERROR, XP_ERROR_NUMBER,
      NULL, 0, __LINE__,
      multibyteErrorMessage,
      SRV_NULLTERM);
  srv_senddone(srvproc, (SRV_DONE_ERROR | SRV_DONE_MOR
  return 1;
}
```
Extended Stored Procedures Programmer's Reference

Microsoft® Open Data Services provides a server-based application programming interface (API) for extending Microsoft SQL Server[™] 2000 functionality. The API consists of C and C++ functions and macros used to build applications.

srv_alloc

Allocates memory dynamically.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM version 7.0 for backward compatibility.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

void * srv_alloc (DBINT size);

Arguments

size

Specifies the number of bytes to allocate.

Returns

A pointer to the newly allocated space. If *size* bytes cannot be allocated, a null pointer is returned.

Remarks

Use the **srv_free** function to free memory allocated by **srv_alloc**. The **srv_alloc** function is equivalent to the Microsoft Win32® **GlobalAlloc** function. Normal Win32 or C run-time memory management functions can be used in an Open Data Services application.

srv_convert

Changes data from one data type to another.

Syntax

int srv_convert (SRV_PROC * srvproc, int srctype, void * src, DBINT srclen, int desttype, void * dest, DBINT destlen);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains all the control information that Open Data Services uses to manage communications and data between the Open Data Services server application and the client. If the *srvproc* handle is supplied, it is passed to the Open Data Services error handler function when an error occurs.

srctype

Specifies the data type of the data to be converted. This parameter can be any of the Open Data Services data types.

src

Is a pointer to the data to be converted. This parameter can be any of the Open Data Services data types.

srclen

Specifies the length, in bytes, of the data to be converted. If *srclen* is 0, **srv_convert** places a null value in the destination variable. Unless it is 0, this parameter is ignored for fixed-length data types, in which case the source

data is assumed to be NULL. For data of the SRVCHAR data type, a length of -1 indicates the string is null-terminated.

desttype

Specifies the data type to convert the source to. This parameter can be any of the Open Data Services data types.

dest

Is a pointer to the destination variable that receives converted data. If this pointer is NULL, **srv_convert** calls the user-supplied error handler (if any) and returns -1.

If *desttype* is SRVDECIMAL or SRVNUMERIC, the *dest* parameter must be a pointer to a DBNUMERIC or DBDECIMAL structure with the precision and scale fields of the structure already set to the desired values. You can use DEFAULTPRECISION to specify a default precision, and DEFAULTSCALE to specify a default scale.

destlen

Specifies the length, in bytes, of the destination variable. This parameter is ignored for fixed-length data types. For a destination variable of type SRVCHAR, the value of *destlen* must be the total length of the destination buffer space. A length of -1 for a destination variable of type SRVCHAR or SRVBINARY indicates there is sufficient space available. For a destination variable of type *srvchar*, a length of -1 causes the character string to be null-terminated.

Returns

The length of the converted data, in bytes, if the data type conversion succeeds. When **srv_convert** encounters a request for a conversion it does not support, it calls the developer-supplied error handler (if any), sets a global error number, and returns -1.

Remarks

The **srv_willconvert** function determines whether a particular conversion is allowed.

Converting to the approximate numeric data types SRVFLT4 or SRVFLT8 can result in some loss of precision. Converting from the approximate numeric data types SRVFLT4 or SRVFLT8 to SRVCHAR or SRVTEXT can also result in some loss of precision.

Converting to SRVFLT*x*, SRVINT*x*, SRVMONEY, SRVMONEY4, SRVDECIMAL, or SRVNUMERIC can result in overflow if the number is larger than the destination's maximum value, or in underflow if the number is smaller than the destination's minimum value. If overflow occurs when converting to SRVCHAR or SRVTEXT, the first character of the resulting value contains an asterisk (*) to indicate the error.

When converting SRVCHAR to SRVBINARY, **srv_convert** interprets SRVCHAR as hexadecimal, whether or not the string contains a leading 0. When converting SRVBINARY to SRVCHAR, **srv_convert** creates a hexadecimal string without a leading 0. In all other cases, a conversion to or from the SRVBINARY data type is a straight bit-copy.

In certain cases, it can be useful to convert a data type to itself. For example, converting SRVCHAR to SRVCHAR with a *destlen* of -1 adds a null terminator to a string.

For a description of data types and Open Data Services data type conversions, see <u>Data Types</u>.

The **srv_convert** function can fail for several reasons:

- The requested conversion is not available.
- The conversion resulted in truncation, overflow, or loss of precision in the destination variable.
- A syntax error occurred while converting a character string to a numeric data type.

See Also

srv_setutype

srv_willconvert

srv_describe

Defines the column name and source and destination data types for a specific column in a row.

Syntax

int srv_describe (SRV_PROC * srvproc, int colnumber, DBCHAR * column_name, int namelen, DBINT desttype, DBINT destlen, DBINT srctype, DBINT srctype, DBINT srclen, void * srcdata);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the client sending the row). The structure contains all the information that the ODS Library uses to manage communications and data between the Open Data Services server application and the client.

colnumber

Is currently not supported. Columns must be described in order. All columns must be described before **srv_sendrow** is called.

column_name

Specifies the name of the column to which the data belongs. This parameter can be NULL because a column is not required to have a name.

namelen

Specifies the length, in bytes, of *column_name*. If *namelen* is

SRV_NULLTERM, then *column_name* must be null-terminated.

desttype

Specifies the data type of the destination row column (the data type sent to the client). The data type must be specified even if the data is NULL (see <u>Data Types</u>).

destlen

Specifies the length, in bytes, of the data to be sent to the client. For fixed-length data types that do not allow null values, *destlen* is ignored. For variable-length data types and fixed-length data types that allow null values, *destlen* specifies the maximum length the destination data can be.

srctype

Specifies the data type of the source data.

srclen

Specifies the length, in bytes, of the source data. This value is ignored for fixed-length data types.

srcdata

Provides the source data address for a particular column. When **srv_sendrow** is called, it looks for the data for *colnumber* at *srcdata*. Therefore should not be freed before a call to **srv_sendrow**. The source data address can be changed between calls to **srv_sendrow** by using **srv_setcoldata**. Memory allocated for *srcdata* should not be freed until **srv_senddone** is called.

If *desttype* is SRVDECIMAL or SRVNUMERIC, the *srcdata* parameter must be a pointer to a DBNUMERIC or DBDECIMAL structure with the precision and scale fields of the structure already set to the values you want. You can use DEFAULTPRECISION to specify a default precision, and DEFAULTSCALE to specify a default scale.

Returns

The number of the column described. The first column is column 1. If an error occurs, returns 0.

Remarks

The **srv_describe** function must be called once for each column in the row before the first call to **srv_sendrow**. The columns of a row can be described in any order.

To change the location and length of the source data in column rows before the complete result set has been sent, use **srv_setcoldata** and **srv_setcollen**, respectively.

For a description of data types and Open Data Services data type conversions, see <u>Data Types</u>.

If the column name in your application is in Unicode, you need to convert it to the multibyte code page of the server before calling **srv_describe**. For more information, see <u>Unicode Data and Server Code Pages</u>.

See Also

srv_sendrow srv_setutype srv_setcoldata

srv_getbindtoken

Obtains a bind token so an extended stored procedure session can share a common transaction lock space with the client session that invoked the extended stored procedure, instead of opening a separate session.

Syntax

```
int srv_getbindtoken ( SRV_PROC* srvproc,
char* bindtoken );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The *srvproc* parameter contains information that the ODS Library uses to manage communication and data between the application and client.

bindtoken

Is a pointer to a buffer where the bind token will be copied. The bind token is represented as a null-terminated string. The buffer you specify should be 255 bytes in length.

Returns

SUCCEED or FAIL.

Remarks

To bind an extended stored procedure session to the client session that called it so they share the same transaction lock space

 The extended stored procedure calls svr_getbindtoken. This uses the existing bind token of the client session if the session is already bound. If not, srv_getbindtoken creates a new bind token and binds the client session to the token. srv_getbindtoken then returns the bind token in the *bindtoken* parameter.

- 2. The extended stored procedure uses the bind token with **sp_bindsession** to bind an extended stored procedure session to the client session. Multiple extended stored procedure sessions can be bound to a client session.
- 3. A bound session is unbound when the external stored procedure returns or when **sp_bindsession** is called with an empty string.

Note Only one bound session at a time can have access to a shared connection. If one session is currently executing a statement at the server or has results pending from the server, no other sessions sharing the same bound connection can gain access to the server until the current session has finished executing the current statement. If a session attempts to gain access to the connection while the server is busy, an error is returned to the conflicting session indicating the connection is in use and the session should retry later.

See Also

<u>sp_bindsession</u>

sp_getbindtoken

srv_message_handler

Calls the installed Open Data Services message handler. This function is usually used to call Microsoft® SQL ServerTM from an extended stored procedure to log an error (defined by the extended stored procedure) in the SQL Server error log file or the Microsoft Windows® application log.

Syntax

int srv_message_handler (SRV_PROC * srvproc, int errornum, BYTE severity, BYTE state, int oserrnum, char * errtext, int errtextlen, char * oserrtext, int oserrtext, int oserrtextlen);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The *srvproc* parameter contains information that the ODS Library uses to manage communication and data between the application and the client.

errornum

Is an error number defined by the extended stored procedure. This number must be from 50,001 through 2,147,483,647.

severity

Is a standard SQL Server severity value for the error. This number must be from 0 through 24.

state

Is a SQL Server state value for the error.

oserrnum

Is the operating-system error number. This argument is ignored.

errtext

Is the description of the extended stored procedure error *errornum*.

errtextlen

Is the length of the extended stored procedure error string *errtext*.

oserrtext

Is the description of the operating-system error *oserrnum*. This argument is ignored.

oserrtextlen

Is the length of the operating-system error string *oserrtext*.

Returns

SUCCEED or FAIL.

Remarks

The **srv_message_handler** function enables an extended stored procedure to integrate with the centralized error logging and reporting features of SQL Server. SQL Server alerts can be established for events from extended stored procedures, and SQL Server Agent will monitor for these alert conditions.

If the error message is longer, it is truncated to 412 bytes.

srv_paramdata

Returns the value of a remote stored procedure call parameter.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM 2000 for backward compatibility.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3)</u>.

Syntax

```
void * srv_paramdata ( SRV_PROC * srvproc,
int n );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure call). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

n

Is the number of the parameter. The first parameter is number 1.

Returns

A pointer to the parameter value. If the *n*th parameter is NULL, there is no *n*th parameter, or there is no remote stored procedure, returns NULL. If the parameter value is a string, it might not be null-terminated. Use **srv_paramlen** to determine the length of the string.

This function returns the following values, if the parameter is one of the SQL Server data types. Pointer data includes whether the pointer for the data type is valid (VP), NULL, or not applicable (N/A), and the contents of the data pointed

to.

New data types	Input data length			
	NULL	ZERO	>=255	<255
BITN	VP	VP	N/A	N/A
	NULL	NULL		
BIGVARCHAR	NULL	VP	VP	VP
	N/A	NULL	255 chars*	actual data
BIGCHAR	NULL	VP	VP	VP
	N/A	255 spaces	255 chars*	actual data
				+ padding
				(up to 255)
BIGBINARY	NULL	VP	VP	VP
	N/A	255 0x00	255 bytes	actual data
				+ padding
				(up to 255)
BIGVARBINARY	NULL	VP	VP	VP
	N/A	0x00	255 bytes	actual data
NCHAR	NULL	VP	VP	VP
	N/A	255 spaces	255 chars*	actual data
				+ padding
				(up to 255)
NVARCHAR	NULL	VP	VP	VP
	N/A	NULL	255 chars*	actual data
NTEXT	N/A	N/A	N/A	N/A

* data is not null-terminated; no warning is issued on truncation for data >255 characters.

Remarks

If you know the parameter name, you can use **srv_paramnumber** to get the parameter number. To determine whether a parameter is NULL, use **srv_paramlen**.

When a remote stored procedure call is made with parameters, the parameters can be passed by name or by position (unnamed). If the remote stored procedure call is made with some parameters passed by name and some passed by position, an error occurs. If an error occurs, the SRV_RPC handler is still called, but it

appears as if there were no parameters and **srv_rpcparams** returns 0.

See Also

srv_rpcparams

srv_paraminfo

Returns information about a parameter.

IMPORTANT This function supersedes the following Open Data Services functions: <u>srv_paramtype</u>, <u>srv_paramlen</u>, <u>srv_parammaxlen</u>, and <u>srv_paramdata</u>. **srv_paraminfo** supports the new <u>Data Types</u> and zero-length data.

Syntax

int srv_paraminfo (SRV_PROC * srvproc, int n, BYTE * pbType, ULONG * pcbMaxLen, ULONG * pcbActualLen, BYTE * pbData, BOOL * pfNull);

Arguments

srvproc

A handle for a client connection.

n

The ordinal number of the parameter to be set. The first parameter is 1.

pbType

The data type of the parameter.

pcbMaxLen

Pointer to the maximum length of the parameter.

pcbActualLen

Pointer to the actual length of the parameter. A value of 0 (**pcbActualLen* == 0) signifies zero-length data if **pfNull* is set to FALSE.

pbData

Pointer to the buffer for parameter data. If *pbData* is not NULL, Open Data Services writes **pcbActualLen* bytes of data to **pbData*. If *pbData* is NULL, no data is written to **pbData* but the function returns **pbType*, **pcbMaxLen*, **pcbActualLen*, and **pfNull*. The memory for this buffer must be managed by the Open Data Services application.

pfNull

Pointer to a null flag. **pfNull* is set to TRUE if the value of the parameter is NULL.

Returns

If the parameter information was successfully obtained, SUCCEED is returned; otherwise, FAIL. FAIL is returned when there is no current remote stored procedure and when there is no *n*th remote stored procedure parameter.

srv_paramlen

Returns the data length of a remote stored procedure call parameter.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM 2000 for backward compatibility. It has been superseded by the **srv_paraminfo** function.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

int srv_paramlen (SRV_PROC * srvproc, int n);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure call). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

n

Indicates the number of the parameter. The first parameter is 1.

Returns

The actual length, in bytes, of the parameter data. If there is no *n*th parameter or there is no remote stored procedure, returns -1. If the *n*th parameter is NULL, returns 0.

This function returns the following values, if the parameter is one of the following SQL Server data types.

New data types	Input data length			
	NULL	ZERO	>=255	<255
BITN	1	1	N/A	N/A
BIGVARCHAR	0	1	255	actual len*
BIGCHAR	0	255	255	255
BIGBINARY	0	255	255	255
BIGVARBINARY	0	1	255	actual len*
NCHAR	0	255	255	255
NVARCHAR	0	1	255	actual len*
NTEXT	-1	-1	-1	-1

* actual len = Length of multibyte character string (cch)

Remarks

Each remote stored procedure parameter has an actual and a maximum data length. For standard fixed-length data types that do not allow null values, the actual and maximum lengths are the same. For variable-length data types, the lengths can vary. For example, a parameter declared as **varchar(30)** can have data that is only 10 bytes long. The parameter's actual length is 10 and its maximum length is 30. The **srv_paramlen** function gets the actual data length, in bytes, of a remote stored procedure. To obtain the maximum data length of a parameter, use **srv_parammaxlen**.

When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored procedure call is made with some parameters passed by name and some passed by position, an error occurs. The SRV_RPC handler is still called, but it appears as if there were no parameters and **srv_rpcparams** returns 0.

See Also

<u>srv_paraminfo</u>

<u>srv_rpcparams</u>

srv_parammaxlen

Returns the maximum data length of a remote stored procedure call parameter.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM 2000 for backward compatibility. It has been superseded by the **srv_paraminfo** function.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

int srv_parammaxlen (SRV_PROC * srvproc, int n);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure call). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

n

Indicates the number of the parameter. The first parameter is 1.

Returns

The maximum length, in bytes, of the parameter data. If there is no *n*th parameter or if there is no remote stored procedure, returns -1.

This function returns the following values, if the parameter is one of the following SQL Server data types.

New data types	Input data length			

	NULL	ZERO	>=255	<255
BITN	1	1	N/A	N/A
BIGVARCHAR	255	255	255	255
BIGCHAR	255	255	255	255
BIGBINARY	255	255	255	255
BIGVARBINARY	255	255	255	255
NCHAR	255	255	255	255
NVARCHAR	255	255	255	255
NTEXT	-1	-1	-1	-1

Remarks

Each remote stored procedure parameter has an actual and a maximum data length. For standard fixed-length data types that do not allow null values, the actual and maximum lengths are the same. For variable-length data types, the lengths can vary. For example, a parameter declared as **varchar(30)** can have data that is only 10 bytes long. The parameter's actual length is 10 and its maximum length is 30. The **srv_parammaxlen** function gets the maximum data length of a remote stored procedure. To obtain the actual length of a parameter, use **srv_paramlen**.

When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored procedure call is made with some parameters passed by name and some passed by position, an error occurs. The SRV_RPC handler is still called, but it appears as if there were no parameters, and **srv_rpcparams** returns 0.

See Also

srv_paraminfo

<u>srv_rpcparams</u>

srv_paramname

Returns the name of a remote stored procedure call parameter.

Syntax

DBCHAR * srv_paramname (SRV_PROC srvproc,

int n ,
int len);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

n

Indicates the number of the parameter. The first parameter is 1.

len

Provides a pointer to an **int** variable that contains the length, in bytes, of the parameter name. If *len* is NULL, the length of the remote stored procedure parameter name is not returned.

Returns

A pointer to a null-terminated character string that contains the parameter name. The length of the parameter name is stored in *len*. If there is no *n*th parameter or no remote stored procedure, returns NULL, *len* is set to -1, and an informational error message is sent. If the parameter name is NULL, *len* is set to 0 and a null-terminated empty string is returned.

Remarks

This function gets the name of a remote stored procedure call parameter. When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored procedure call is made with some parameters passed by name and some passed by position, an error occurs. The SRV_RPC handler is still called, but it appears as if there were no parameters, and **srv_rpcparams** returns 0.

See Also

srv_rpcparams

srv_paramnumber

Returns the number of a remote stored procedure call parameter.

Syntax

```
int srv_paramnumber ( SRV_PROC * srvproc,
DBCHAR * name,
int namelen );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure call). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

name

Is a pointer to the parameter name.

namelen

Is the length of *name*. If *name* is null-terminated, set *namelen* to SRV_NULLTERM.

Returns

The parameter number of the named parameter. The first parameter is 1. If there is no parameter named *name* or no remote stored procedure, 0 is returned and a message is generated.

Remarks

When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored
procedure call is made with some parameters passed by name and some passed by position, an error occurs. The SRV_RPC handler is still called, but it appears as if there were no parameters, and **srv_rpcparams** returns 0.

See Also

srv_rpcparams

srv_paramset

Sets the value of a remote stored procedure call return parameter.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM 2000 for backward compatibility. It has been superseded by the **srv_paramsetoutput** function.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

int srv_paramset (SRV_PROC * srvproc, int n , void * data, int len);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure call). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

n

Indicates the number of the parameter to set. The first parameter is 1.

data

Is a pointer to the data value to be sent back to the client as the remote stored procedure return parameter.

len

Specifies the actual length of the data to be returned. If the data type of the parameter is of a constant length and does not allow null values (for

example, *srvbit* or *srvint1*), *len* is ignored.

Returns

SUCCEED if the parameter value was successfully set; otherwise, FAIL. FAIL is returned when there is no current remote stored procedure, when there is no *n*th remote stored procedure parameter, when the parameter is not a return parameter, and when the *len* argument is not legal.

If *len* is 0, returns NULL. Setting *len* to 0 is the only way to return NULL to the client.

This function returns the following values, if the parameter is one of SQL Server data types.

New data types	Return data	length		
	NULL	ZERO	>=255	<255
BITN	len = 0 data = IG RET = 0	N/A	N/A	N/A
BIGVARCHAR	len = 0	len = IG	len = max8k	len = <8k
	data = IG	data = IG	data = valid	data = valid
	RET = 1	RET = 0	RET = 0	RET = 1
BIGCHAR	len = 0	len = IG	len = max8k	len = <8k
	data = IG	data = IG	data = valid	data = valid
	RET = 1	RET = 0	RET = 0	RET = 1
BIGBINARY	len = 0	len = IG	len = max8k	len = <8k
	data = IG	data = IG	data = valid	data = valid
	RET = 1	RET = 0	RET = 0	RET = 1
BIGVARBINARY	len = 0	len = IG	len = max8k	len = <8k
	data = IG	data = IG	data = valid	data = valid
	RET = 1	RET = 0	RET = 0	RET = 1
NCHAR	len = 0	len = IG	len = max8k	len = <8k
	data = IG	data = IG	data = valid	data = valid
	RET = 1	RET = 0	RET = 0	RET = 1
NVARCHAR	len = 0	len = IG	len = max8k	len = <8k
	data = IG	data = IG	data = valid	data = valid

	RET = 1	RET = 0	RET = 0	RET = 1
NTEXT	len = IG	len = IG	len = IG	len = IG
	data = IG	data = IG	data = IG	data = IG
	RET = 0	RET = 0	RET = 0	RET = 0
RET = Return value of srv_paramset				
IG = Value will be ignored				
valid = Any valid pointer to data				

Remarks

Parameters contain data passed between clients and the Open Data Services server application with remote stored procedures. The client can specify certain parameters as return parameters. These return parameters can contain values that the Open Data Services server application passes back to the client. Using return parameters is analogous to passing parameters by reference.

You cannot set the return value for a parameter that wasn't invoked as a return parameter. You can use **srv_paramstatus** to determine how the parameter was invoked.

This function sets the return value for a parameter but it does not actually send the return value to the client. All return parameters, whether their return values have been set with **srv_paramset** or not, are automatically sent to the client when **srv_senddone** is called with the status flag SRV_DONE_FINAL set.

When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored procedure call is made with some parameters passed by name and some passed by position, an error occurs. The SRV_RPC handler is still called, but it appears as if there were no parameters, and **srv_rpcparams** returns 0.

See Also

srv_paramsetoutput

srv_paramsetoutput

Sets the value of a return parameter.

IMPORTANT This function supersedes the **srv_paramset** function. **srv_paramsetoutput** supports the new Open Data Services data types and zero-length data.

Syntax

int srv_paramsetoutput (SRV_PROC * srvproc, int n, BYTE * pbData, ULONG cbLen, BOOL fNull);

Arguments

srvproc

Is a handle for a client connection.

n

Is the ordinal number of the parameter to be set. The first parameter is 1.

pbData

Is a pointer to the data value to be sent back to the client as a procedure return parameter.

cbLen

Is the actual length of the data to be returned. If the data type of the parameter specifies values of a constant length and does not allow null values (for example, *srvbit or srvint1*), *cbLen* is ignored. A value of 0 signifies zero-length data if *fNull* is FALSE.

fNull

Is a flag indicating whether the value of the return parameter is NULL. Set

this flag to TRUE if the parameter should be set to NULL. The default value is FALSE. If *fNull* is set to TRUE, *cbLen* should be set to 0 or the function will fail.

Returns

If the parameter information was successfully set, SUCCEED is returned; otherwise, FAIL. FAIL is returned when there is no current remote stored procedure, when the parameter is not a return parameter, or when the *cbLen* argument is invalid.

srv_paramstatus

Returns the status of a particular remote stored procedure call parameter.

Syntax

```
int srv_paramstatus ( SRV_PROC * srvproc,
int n );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure call). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

n

Indicates the number of the parameter. The first parameter is number 1.

Returns

An **int** that contains status flags for the parameter. Currently, there is only one flag: If bit 0 is set to 1, the parameter is a return parameter. If there is no *n*th parameter or if there is no remote stored procedure, returns -1.

Remarks

This routine returns the status flags for a remote stored procedure call parameter.

Parameters contain data passed between clients and the Open Data Services server application with remote stored procedures. The client can specify certain parameters as return parameters. These return parameters can contain values that the Open Data Services server application passes back to the client.

Currently, the only status flag is one that indicates whether the parameter is a

return parameter.

Open Data Services can use **srv_paramset** to set the value of a return parameter.

When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored procedure call is made with some parameters passed by name and some passed by position, an error occurs. If an error occurs, the SRV_RPC handler is still called, but it appears as if there were no parameters, and **srv_rpcparams** returns 0.

See Also

<u>srv_rpcparams</u>

srv_paramtype

Returns the data type of a remote stored procedure call parameter.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM 2000 for backward compatibility. It has been superseded by the **srv_paraminfo** function.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

```
int srv_paramtype ( SRV_PROC * srvproc,
int n );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure call). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

n

Indicates the number of the parameter. The first parameter is 1.

Returns

A token value for the data type of the parameter. You can get a readable description for the value by using **srv_symbol**. For information about a list of data types, see <u>Data Types</u>. If there is no *n*th parameter or if there is no remote stored procedure, returns - 1.

This function returns the following values, if the parameter is one of the SQL Server data types.

New data types	Return value
BITN	SRVBIT
BIGVARCHAR	VARCHAR
BIGCHAR	CHAR
BIGBINARY	BINARY
BIGVARBINARY	VARBINARY
NCHAR	CHAR
NVARCHAR	VARCHAR
NTEXT	-1

Remarks

When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored procedure call is made with some parameters passed by name and some passed by position, an error occurs. The SRV_RPC handler is still called, but it appears as if there were no parameters and **srv_rpcparams** returns 0.

See Also

srv_paraminfo

srv_rpcparams

srv_pfield

Returns information about a database connection.

Syntax

DBCHAR * srv_pfield (SRV_PROC * srvproc, int field, int * len);

Arguments

srvproc

Pointer identifying a database connection.

field

Specifies data on the connection to return.

Value	Returns
SRV_APPLNAME	The application name provided by the
	client when it established the
	connection.
SRV_BCPFLAG	A flag that is TRUE if the client is
	preparing for a bulk copy operation;
	otherwise, FALSE.
SRV_CLIB	The name of the library that enables
	the client to talk to a server.
SRV_CPID	The client process ID on the client
	source computer.
SRV_HOST	The name of the client's machine
	provided by the client when it
	established the connection.
SRV_LIBVERS	The version of the client library.
SRV_LSECURE	A flag. TRUE if connection used
	integrated security to login.

SRV_NETWORK_MODULE	The name of the Net-Library DLL used by the connection.
SRV_NETWORK_VERSION	The version of the Net-Library DLL used used by the connection.
SRV_NETWORK_CONNECTION	The connection string passed to the Net-Library DLL used for the current <i>srvproc</i> connection.
SRV_PIPEHANDLE	A string containing the pipe handle of a connected client, or NULL if the client is connected on a network that does not use named pipes. To use this handle as a valid pipe handle with Microsoft® Windows NT® 4.0, convert this string to an integer.
SRV_PWD	If the connection used standard security to login, the password it provided.
SRV_RMTSERVER	The server from which the client process is logged in. If the login is from a client, this value is an empty string.
SRV_ROWSENT	The number of rows already sent by <i>srvproc</i> for the current set of results.
SRV_SPID	The server thread ID of the <i>srvproc</i> . For extended stored procedures, this value is the same as the kpid column of sysprocesses , and it can change over time.
SRV_SPROC_CODEPAGE	Codepage that the server uses to interpret multbyte data.
SRV_STATUS	The current status of <i>srvproc</i> .
SRV_TDS	The version of the tabular data stream (TDS) used by the client. Use SRV_TDSVERSION instead of this value.
SRV_TYPE	The connection type of <i>srvproc</i> . If

	"server" is returned, <i>srvproc</i> is from an instance of SQL Server. If "client" is returned, <i>srvproc</i> is from a DB-Library or ODBC client.
SRV_USER	The user name of the connection.

len

Is a pointer to an **int** variable that contains the length of the returned *field* value. If *len* is NULL, the length of the string is not returned.

Returns

A pointer to a null-terminated string containing the current value for the specified field in the SRV_PROC structure. If the field is empty, a valid pointer to an empty string is returned and *len* contains 0. If the field is unknown, NULL is returned and *len* contains the value -1.

srv_pfieldex

Returns a pointer to data containing the requested SRV_PROC field.

Syntax

void *sql_pfieldex(SRV_PROC * srvproc, int field, int * len);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

field

Specifies the *srvproc* field to return.

Field	Description	Return-type
SRV_MSGLCID	Current session message LCID.	ULONG*
	Instance name (if named); otherwise, returns NULL.	WCHAR*

len

Is a pointer to an **int** variable that contains the length of the returned *field* value in bytes. If *len* is NULL, the length is not returned. When NULL is returned **len* is set to 0.

Returns

A pointer to data whose type depends on *field*. NULL is returned when *len* is NULL or *srvproc* is NULL. If the *field* is unknown, NULL is returned. When NULL is returned **len* is set to 0.

srv_rpcdb

Returns the database name component for the current remote stored procedure.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM version 7.0 for backward compatibility.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

DBCHAR * srv_rpcdb (SRV_PROC srvproc, int * len);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

len

Is a pointer to an **int** variable that receives the length of the database name. If *len* is NULL, the length of the database name is not returned.

Returns

A DBCHAR pointer to the null-terminated string for the database name part of the current remote stored procedure. If there is no current remote stored procedure, NULL is returned and the *len* parameter is set to - 1.

Remarks

This function returns only the database component of the remote stored procedure object name. It does not include the optional specifiers for owner,

remote stored procedure name, and remote stored procedure number.

srv_rpcname

Returns the procedure name component for the current remote stored procedure.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM version 7.0 for backward compatibility.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

```
DBCHAR * srv_rpcname ( SRV_PROC * srvproc,
int * len );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

len

Is a pointer to an integer variable that receives the length of the database name. If *len* is NULL, the length of the remote stored procedure name is not returned.

Returns

A DBCHAR pointer to the null-terminated string for the remote stored procedure name component of the current remote stored procedure. If there is not a current remote stored procedure, NULL is returned and *len* is set to -1.

Remarks

This function returns only the name of the remote stored procedure. It does not include the optional specifiers for owner, database name, and remote stored procedure number.

Because it is valid to call **srv_rpcname** when there is not a remote stored procedure (no informational error occurs), this function provides a method for determining whether a remote stored procedure exists.

srv_rpcnumber

Returns the number component for the current remote stored procedure call.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM version 7.0 for backward compatibility.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3</u>).

Syntax

int srv_rpcnumber (SRV_PROC * srvproc)

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

Returns

The number component for the current remote stored procedure. If the client does not use a number component when running the remote stored procedure or if there is no current remote stored procedure, returns - 1.

Remarks

This function returns only the number component of the remote stored procedure. It does not include the optional specifiers for owner, remote stored procedure name, and database name.

srv_rpcoptions

Returns run-time options for the current remote stored procedure.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft® SQL ServerTM version 7.0 for backward compatibility.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3)</u>.

Syntax

DBUSMALLINT srv_rpcoptions (SRV_PROC * *srvproc*);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure). The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

Returns

A bitmap that contains the run-time flags joined in a logical OR for the current remote stored procedure. If there is not a current remote stored procedure, 0 is returned and a message is generated.

Remarks

The following table describes each run-time flag.

Run-time flag	Description
SRV_NOMETADATA	The client has requested results without meta data
	information. This flag is only used when the client
	is communicating with an instance of SQL Server.

	An Open Data Services application cannot omit meta data information.
SRV_RECOMPILE	The client has requested to recompile the remote stored procedure before executing it. This flag may not apply to an Open Data Services application.

srv_rpcowner

Returns the owner component for the current remote stored procedure.

IMPORTANT This Open Data Services function or macro is only supported in Microsoft[®] SQL Server[™] version 7.0 for backward compatibility.

For more information about Open Data Services functions or macros supported for backward compatibility, see <u>Open Data Services (Level 3)</u>.

Syntax

```
DBCHAR * srv_rpcowner (SRV_PROC * srvproc, int * len );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

len

Is a pointer to an integer variable that receives the length of the owner name. The parameter *len* can be NULL, in which case the length of the owner component is not returned.

Returns

A DBCHAR pointer to the null-terminated owner component for the current remote stored procedure. If there is no current remote stored procedure, NULL is returned and *len* is set to -1.

Remarks

This function returns only the owner component of the remote stored procedure. It does not include the optional specifiers for name, remote stored procedure name, and remote stored procedure number.

srv_rpcparams

Returns the number of parameters for the current remote stored procedure.

Syntax

int srv_rpcparams (SRV_PROC * srvproc);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the remote stored procedure). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

Returns

The number of parameters in the remote stored procedure. If there are no parameters in the remote stored procedure or if there is not a current remote stored procedure, -1 is returned and an information error occurs.

Remarks

This function returns the number of parameters in the current remote stored procedure. It is usually called from the remote stored procedure.

When a remote stored procedure call is made with parameters, the parameters can be passed either by name or by position (unnamed). If the remote stored procedure call was made with some parameters passed by name and some passed by position, an error occurs. When this error occurs, the remote stored procedure handler is called, but it does not receive the parameters and **srv_rpcparams** returns 0.
srv_senddone

Sends a result completion message to the client.

Syntax

int srv_senddone (SRV_PROC * srvproc, DBUSMALLINT status, DBUSMALLINT info, DBINT count);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the language request). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

status

Is a 2-byte field for various *status* flags. Multiple flags can be set by using the AND and OR logical operators with *status* flag values. The following table lists possible *status* flags.

Status flag	Description
SRV_DONE_FINAL	The current set of results is the final set of
	results.
SRV_DONE_MORE	The current set of results is not the final set
	of results.
SRV_DONE_COUNT	The <i>count</i> parameter contains a valid
	count.
SRV_DONE_ERROR	The current client command received an
	error.
SRV_DONE_RPC_IN_BATCH	The current set of results is the last set of

results for the currently executing remote
stored procedure. This value is used when
multiple remote stored procedures are
executed in a single batch.

info

Is a reserved, 2-byte field. Set this value to 0.

count

Is a 4-byte field used to indicate a count for the current result set. If the SRV_DONE_COUNT flag is set in the *status* field, *count* holds a valid count.

Returns

SUCCEED or FAIL

Remarks

A client request can cause the server to execute a number of commands and to return a number of result sets. For each result set, **srv_senddone** must return a result completion message to the client.

To indicate that there are more results for the current request, the Open Data Services server application must set the *status* field to SRV_DONE_MORE. To indicate that there are no more results for the current request, the Open Data Services server application must set the *status* field to SRV_DONE_FINAL. However, when an application function is installed as a Microsoft® SQL ServerTM extended stored procedure instead of as a separate Open Data Services server application, use SRV_DONE_MORE. With extended stored procedures, SQL Server itself sends SRV_DONE_FINAL.

The *count* field indicates the number of rows affected by a command. If the *count* field contains a count, the SRV_DONE_COUNT flag should be set in the *status* field. This setting allows the client to distinguish between a *count* value of 0 and an unused *count* field.

Do not call **srv_senddone** from the SRV_CONNECT handler.

srv_sendmsg

Sends a message to the client.

Syntax

int srv_sendmsg (SRV_PROC * srvproc, int msgtype, DBINT msgnum, DBTINYINT class, DBTINYINT state, DBCHAR * rpcname, int rpcnamelen, DBUSMALLINT linenum, DBCHAR * message, int msglen);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the language request). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

msgtype

Is either SRV_MSG_INFO or SRV_MSG_ERROR, depending on whether the server is sending an informational or error message.

msgnum

Is a 4-byte message number.

class

Specifies the error severity. A severity less than or equal to 10 is considered an informational message.

state

Provides the error state number for the current message. The error state number provides information about the context of the error. Valid state numbers are from 1 through 127.

rpcname

Is currently not supported.

rpcnamelen

Is currently not supported.

linenum

Is the line number in the language command batch where the message applies. Line numbers start at 1. If *linenum* does not apply to the message, set to 0.

message

Is a pointer to the character string to be sent to the client.

msglen

Specifies the length, in bytes, of *message*. If *message* is null-terminated, set *msglen* to SRV_NULLTERM.

Returns

SUCCEED or FAIL

Remarks

This function sends error or informational messages to the client. It is called once for each message to be sent.

Messages sent with **srv_sendmsg** can be sent to the client in any order before or after all rows (if any) have been sent with **srv_sendrow**. All messages, if any, must be sent to the client before the completion status is sent with **srv_senddone**.

If the error message in your application is in unicode, you need to convert it to

the multibyte code page of the server before calling **srv_sendmsg** since message is defined as DBCHAR*.

If the error message in your application is in unicode, convert it to the multibyte code page of the server before calling **srv_sendmsg** since message is defined as DBCHAR*. For more information see <u>Unicode Data and Server Code Pages</u>.

srv_sendrow

Transmits a row of data to the client.

Syntax

int srv_sendrow (SRV_PROC * srvproc);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection (in this case, the handle that received the language request). The structure contains information that the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

Returns

SUCCEED or FAIL.

Remarks

The **srv_sendrow** function is called once for each row sent to the client. All rows must be sent to the client before any messages, status values, or completion statuses are sent with **srv_sendmsg**, **srv_status**, or **srv_senddone**.

Sending a row that has not had all its columns defined with **srv_describe** causes the Open Data Services server application to raise an informational error message and return FAIL to the client. In this case, the row is not sent.

Note Open Data Services does not support sending compute rows to the client. Also, if a row containing **ntext**, **text** or **image** data is sent to the client, the text pointer and text timestamp are not included.

See Also

srv_describe

srv_setcoldata

Specifies the current address for a column's data.

Syntax

```
int srv_setcoldata ( SRV_PROC * srvproc,
int column,
void * data );
```

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

column

Indicates the number of the column the address is being specified for. Columns are numbered beginning with 1.

data

Is a pointer for a column's data. Memory allocated for *data* should not be freed until **srv_senddone** is called.

Returns

SUCCEED or FAIL.

Remarks

Each column of the row must be defined first with **srv_describe**. Column data addresses are initially set with **srv_describe**. If the address of the column data changes, **srv_setcoldata** must be called to specify the new address of the data and **srv_setcoldata** must be called separately for each changed column.

Null data is represented by setting the column's length to 0 with **srv_setcollen**. The data address is then ignored.

See Also

srv_describe

srv_setcollen

Specifies the current data length in bytes of a variable-length column or a column that allows NULL values.

Syntax

int srv_setcollen (SRV_PROC * srvproc, int column, int len);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

column

Indicates the number of the column for which the data length is being specified. Columns are numbered beginning with 1.

len

Indicates the length, in bytes, of the column data. A length of 0 means the column data value is null.

Returns

SUCCEED or FAIL.

Remarks

Each column of the row must first be defined with **srv_describe**. The column data length is set by the last call to **srv_describe** or **srv_setcollen**. If variable-length data (null-terminated data) changes for a row, **srv_setcollen** must be used

to set it to the new length before calling **srv_sendrow**. For a column that allows null values, **srv_describe** must have been called with *desttype* set to a data type that allows nulls (like SRVINTN) and null data is specified by calling **srv_setcollen** with *len* set to 0. Zero length data cannot be specified using Open Data Services API.

Note that when the data type of the column is variable-length, *len* is not checked. This function returns FAIL if called for a fixed-length column.

See Also

srv_describe

srv_setutype

Sets the user-defined data type for a column in a row.

Syntax

int srv_setutype (SRV_PROC * srvproc, int column, DBINT user_type);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

column

Indicates which column to set. Columns are numbered beginning with 1.

user_type

Specifies the user-defined data type code.

Returns

SUCCEED or FAIL. Returns FAIL if the column does not exist.

Remarks

A column has two data types: its actual data type and its user-defined data type. The user-defined data type is used by Microsoft® SQL ServerTM to store the actual user-defined data type of the column (if any) and column description information (such as nullability and updatability) for the column.

The **srv_setutype** function can be called any time that *column* has been defined with **srv_describe** and before the last row has been sent.

See Also

<u>srv_describe</u>

srv_setRPC

This function allows setting individual RPC values (including parameters).

Syntax

void srv_setRPC(SRV_PROC * srvproc, int iItem, ULONG ulValue, Void * pValue);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains all information the ODS Library needs to manage communication and data between the Open Data Services server application and the client.

iItem

Is the RPC value to set (see Remarks for supported values).

ulValue

Is the value to which to set the item (where applicable). If *ulValue* is SRV_NULLTERM, then the RPC value is assumed to be null-terminated.

pValue

Is a pointer to the value to which to set the item (where applicable).

Remarks

This function is used to set individual RPC item values when RPC processing is in progress.

	Applicable	
iItem	parameters	Description

SRV_RPC_ACTIVEFLAG	ulValue	Sets the RPC active flag to <i>ulValue</i> . This flag indicates that an RPC is active.
SRV_RPC_SERVER	ulValue pValue	Sets the RPC server name value. The name is stored at the location pointed to by <i>pValue</i> . The length of the name is stored in <i>ulValue</i> .
SRV_RPC_DATABASE	ulValue pValue	Sets the RPC database name value. The name is stored at the location pointed to by <i>pValue</i> . The length of the name is stored in <i>ulValue</i> .
SRV_RPC_OWNER	ulValue pValue	Sets the RPC owner name value. The name is stored at the location pointed to by <i>pValue</i> . The length of the name is stored in <i>ulValue</i> .
SRV_RPC_OWNER	ulValue pValue	Sets the RPC owner name value. The name is stored at the location pointed to by <i>pValue</i> . The length of the name is stored in <i>ulValue</i> .
SRV_RPC_OWNER	ulValue pValue	Sets the RPC owner name value. The name is stored at the location pointed to by <i>pValue</i> . The length of the name is stored in <i>ulValue</i> .
SRV_RPC_PROCNUMBER	ulValue	Sets the RPC procedure number value.
SRV_RPC_PROCNUMBER	ulValue	Sets the RPC procedure line number value.
SRV_RPC_OPTIONS	ulValue	Sets the RPC option flag. The option flag is 16 bits in size. Bit 0 is set if the RPC is being sent with the recompile

		option. Bit 1 is set if xp_cursor operations are not supposed to return meta data. Bits 2 through 15 are reserved and should be set to zero.
SRV_RPC_NUMPARAMS	ulValue	Specifies how many RPC parameters are contained within this RPC. This value must be set before setting any parameter values.
SRV_RPC_RPCPARAMS	ulValue pValue	Sets the RPC parameter value. The RPC parameter to set is specified in <i>ulValue</i> and the RPC parameter data is contained in the structure, <i>pValue</i> . The RPC parameter structure definition is defined in the structure, SRV_RPCp, which is defined in the Srvstruc.h header file.

srv_willconvert

Determines whether a specific data type conversion is available within the ODS Library.

Syntax

BOOL srv_willconvert (int srctype, int desttype);

Arguments

srctype

Indicates the data type of the data to be converted. This parameter can be any of the Open Data Services data types.

desttype

Indicates the data type to which the source data is converted. This parameter can be any of the Open Data Services data types.

Returns

TRUE if the data type conversion is supported; FALSE if the data type conversion is not supported.

Remarks

For a description of each data type and Open Data Services data type conversions, see <u>Data Types</u>.

See Also

<u>srv_convert</u>

sql_wsendmsg

Sends a Unicode message to the client.

Syntax

Int sql_wsendmsg(SRV_PROC * srvproc, int msgnum, int severity, WCHAR
* message, int msglen);

Arguments

srvproc

Is a pointer to the SRV_PROC structure that is the handle for a particular client connection. The structure contains information the ODS Library uses to manage communication and data between the Open Data Services server application and the client.

msgnum

Is a 4-byte message number.

severity

Specifies the severity of the error. A severity less than or equal to 10 is considered an informational message; otherwise, it is an error.

message

Is a pointer to a Unicode string to be sent to the client.

msglen

Specifies the length, in characters, of *message*. It not support SRV_NULLTERM for msglen like srv_sendmsg)

Returns

SUCCEED or FAIL.

Errors

The following error values are returned by Open Data Services.

Number	Description
17801	Unknown internal error value.
17802	Creation of server event thread failed.
17803	Insufficient memory available.
17804	Invalid ' <i>n</i> bytes' value.
17805	Invalid buffer received from client.
17806	Invalid event specification.
17807	Invalid event '%l!ld!'.
17808	Invalid starting position specified.
17809	Unable to connect. The maximum number of '%1!1d!' users is currently connected.
17810	Unable to set up named pipe.
17811	Requested data conversion does not exist.
17812	Data conversion resulted in overflow.
17813	Attempt to convert data stopped by syntax error in source field.
17814	Invalid function parameter.
17815	No longer waiting for client connections using '%1'.
17816	No active RPC, or parameter value out of range.
17817	No active RPC, or no parameters.
17818	No active RPC, or parameter name not found.
17819	No active RPC.
17820	Invalid data type parameter.
17821	Unable to set up subchannel.
17822	Unable to load ListenOn Net-Library '%1'.
17823	Unable to read from ListenOn connection.
17824	Unable to write to ListenOn connection '%1', loginname '%2', hostname '%3'.
17825	Unable to close ListenOn connection.

17826	Unable to set up ListenOn connection '%1'.
17827	The maximum number of '%1!ld!' remote connections is
	currently in use.
17828	Unable to read from local subchannel named pipe.
17829	Unable to copy buffer to subchannel thread, subchannel
	closed.
17830	A subchannel protocol error has occurred.
17831	Unable to load ListenOn Net-Library '%1' version '%2'. Need
	Net-Library version '%3' or greater.
18732	Unable to read login packet(s).
18733	ListenOn connection '%1' is already in use.
17834	Using '%1' version '%2' to listen on '%3'.
17835	Configured for local access only.
17836	Unable to create IO completion port.

Data Types

To use the Open Data Services data types, include the Srv.h header file in your program. Open Data Services applications also use the same data type definitions as DB-Library functions.

	SQL Server data	
Data type	type	Description
SRVBIGBINARY	binary	binary data type, length 0 to
		8000 bytes.
SRVBIGCHAR	char	character data type, length 0 to 8000 bytes.
SRVBIGVARBINARY	varbinary	Variable-length binary data type, length 0 to 8000 bytes.
SRVBIGVARCHAR	varchar	Variable-length character data type, length 0 to 8000 bytes.
SRVBINARY	binary	binary data type.
SRVBIT	Bit	bit data type.
SRVBITN	bit null	bit data type, null values allowed.
SRVCHAR	char	character data type.
SRVDATETIME	datetime	8-byte datetime data type.
SRVDATETIM4	smalldatetime	4-byte smalldatetime data type.
SRVDATETIMN	datetime null	smalldatetime or datetime
		data type, null values allowed.
SRVDECIMAL	decimal	decimal data type.
SRVDECIMALN	decimal null	decimal data type, null values allowed.
SRVFLT4	real	4-byte real data type.
SRVFLT8	float	8-byte float data type.
SRVFLTN	real float null	real or float data type, null

		values allowed.
SRVIMAGE	image	image data type.
SRVINT1	tinyint	1-byte tinyint data type.
SRVINT2	smallint	2-byte smallint data type.
SRVINT4	Int	4-byte int data type.
SRVINTN	tinyint smallint in	t tinyint, smallint, or int data
	null	type, null values allowed.
SRVMONEY4	smallmoney	4-byte smallmoney data
		type.
SRVMONEY	money	8-byte money data type.
SRVMONEYN	money smallmoney	5 5
	null	type, null values allowed.
SRVNCHAR	nchar	Unicode character data
		type.
SRVNTEXT	ntext	Unicode text data type.
SRVNUMERIC	numeric	numeric data type.
SRVNUMERICN	numeric null	numeric data type, null
		values allowed.
SRVNVARCHAR	nvarchar	Unicode variable-length
		character data type.
SRVTEXT	text	text data type.
SRVVARBINARY	varbinary	Variable-length binary data
		type.
SRVVARCHAR	varchar	Variable-length character
		data type.

Open Data Services Data Type Conversions

The following chart shows conversions allowed for Open Data Services data types.

To: From:	SRVCHAR/SRVVARCHAR	SRVBIGCHAR/SRVBIGVARCHAR	SRVTEXT	SRVNCHAR/SRVNVARCHAR	SRVNTEXT	SRVBINARY/SRVVARBINARY	SRVBIGBINARY/SRVBIGVARBINARY	SRVIMAGE	SRVINT1/SRVINT2	SRVINT4/SRVINTN	SRVFINT8	SRVFLT8/SRVFLTN	SRVFLT4	SRVBIT/SRVBITN	SRVMONEY/SRVMONEYN	SRVMONEY4	SRVDATETIME/SRVDATETIMN	SRVDATETIM4	SRVDECIMAL/SRVDECIMALN	SRVNUMERIC/SRVNUMERICN	srvguid
SRVCHAR/SRVVARCHAR		٠	٠	۰	۰	۰	۰	٠	•	۰	۰	۰	۰	٠	٠	۰	٠	٠	٠	٠	٠
SRVBIGCHAR/SRVBIGVARCHAR	٠		٠	۰	٠	۰	۰	٠	•	٠	٠	۰	۰	٠	•	۰	٠	٠	٠	٠	٠
SRVTEXT	•	٠		۰	٠	٠	٠	٠	٠	٠	٠	۰	۰	٠	٠	•	٠	•	٠	٠	٠
SRVNCHAR/SRVNVARCHAR	۰	٠	٠		٠	۰	۰	٠	٠	•	٠	۰	۰	٠	٠	۰	٠	•	•	٠	۲
SRVNTEXT	۰	٠	٠	۰		۲	۰	٠	٠	۲	۲	۰	٠	٠	٠	۲	۲	٠	•	٠	۰
SRVBINARY/SRVVARBINARY	٠	٠	٠	۲	۲		۲	٠	٠	۲	۲	0	0	٠	٠	۰	٠	٠	•	٠	•
SRVBIGBINARY/SRVBIGVARBINARY	٠	٠	٠	۰	٠	٠		٠	٠	٠	٠	0	0	٠	٠	٠	٠	•	•	٠	٠
SRVIMAGE	٠	٠	٠	۰	٠	٠	٠		٠	٠	٠	0	0	٠	٠	٠	٠	•	•	٠	٠
SRVINT1/SRVINT2	٠	٠	٠	۰	٠	•	۰	٠		٠	٠	۰	۰	٠	٠	٠	٠	٠	٠	٠	0
SRVINT4/SRVINTN	٠	٠	٠	۰	•	٠	٠	٠	٠		٠	٠	٠	٠	٠	۰	٠	•	٠	٠	0
SRVINT8	٠	٠	٠	۲	٠	٠	٠	٠	٠	٠		٠	•	٠	٠	۰	٠	•	•	٠	\bigcirc
SRVFLT8/SRVFLTN	•	٠	•	۰	٠	٠	•	•	٠	•	٠		•	•	•	۰		•	•	٠	0
SRVFLT4	٠	٠	•	۰	٠	٠	٠	٠	٠	•	٠	٠		•	٠	۰	٠	٠	•	٠	0
SRVBIT/SRVBITN	٠	٠	٠	۰	۰	٠	۰	٠	•	۰	۰	٠	٠		٠	۰		٠	٠	٠	0
SRVMONEY/SRVMONEYN	۲	۲	٠	۲	•	٠	۲	٠	٠	٠	٠	٠	•	٠		٠	•	•		•	0
SRVMONEY4	٠	٠	٠	•	•	•	٠	٠	•	•	•	٠	•	٠	•		•	•	•	•	0
SRVDATETIME/SRVDATETIMN	•	•	•	٠		•	•	•	•									•	•		0
SRVDATETIM4	٠	٠	٠	٠		•	٠		•	٠	٠	•	٠	•	٠	٠	٠		•	•	0
SRVDECIMAL/SRVDECIMALN	٠	٠	•	•	•	•	٠		•			٠				•	٠	•		•	0
SRVNUMERIC/SRVNUMERICN	٠	٠	•	٠	•	•	•	٠	•	•	•	•		٠		•	•	•	•		0
SRVGUID	٠	٠	٠	٠	٠	٠	٠	٠	0	0	0	0	0	0	0	0	0	0	0	0	

Conversion

O Conversion not allowed
Sample Extended Stored Procedures

The Open Data Services samples are installed by the Microsoft® SQL Server[™] 2000 setup program in subdirectories under *x*:\Program Files\Microsoft SQL Server\80\Tools\DevTools\Samples\ODS. All of the necessary C source, definition, project, and .sql files for the samples are located in these subdirectories.

Extended stored		Source file	
procedure	DLL	subdirectory	Description
xp_hello	Xp_hello.dll	xp_hello	Basic extended stored procedure that accepts one output parameter and prints one result set.
xp_srv_paraminfo_sample	Xp_param.dll	xp_param	Analyzes parameters from an extended stored procedure call, and posts a result set to the client about each parameter and parameter value.
xp_gettable_odbc	Xp_odbc.dll	xp_odbc	Uses ODBC to open a bound connection to the same instance of SQL Server that called the extended stored procedure. The server returns a result set to the

			extended stored procedure, which passes the result set to the client.
xp_gettable_dblib	Xp_dblib.dll	-	Uses the DB- Library interface to open a bound connection to the same instance of SQL Server that called the extended stored procedure. The server returns a result set to the extended stored procedure, which passes the result set to the client.

To create the extended stored procedure DLLs for these samples, go to the appropriate sample subdirectory. Using Microsoft Visual C++® 6.0, open the .dsw file and compile this program. Perform these steps before you compile:

1. On the **Tools** menu, click **Options**, and then click the **Directories** tab.

2. In the **Show directories for** box, select **Include files** and **Library files**. Ensure these directories (as appropriate) are included and also appear on the top list:

- Include files: ..\Tools\Devtools\Include
- Library files: ..\Tools\Devtools\Lib

See Also

Adding an Extended Stored Procedure to SQL Server

Creating Extended Stored Procedures

Debugging an Extended Stored Procedure

Removing an Extended Stored Procedure from SQL Server

Using Bound Connections

Using xp_hello

This Microsoft® Visual C++® sample shows a basic extended stored procedure that accepts one output parameter and prints one result set.

Default Location

```
x:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\ODS\Xp_hello
```

Running the Sample

To create the extended stored procedure DLLs for these samples, in the appropriate sample subdirectory, open the supplied .dsw file in Microsoft Visual C++.

To run this program

- 1. Build and compile the Xp_hello.dll.
- 2. Place the compiled Xp_hello.dll in the Microsoft SQL Server\80\Tools\Binn directory.
- 3. Start the server.
- 4. Start SQL Query Analyzer, and then run the Xp_hello.sql script.

See Also

Extended Stored Procedure Sample: xp_hello Samples

Using xp_srv_paraminfo_sample

This Microsoft® Visual C++® sample shows how to analyze parameters from an extended stored procedure call, and post a result set to the client about each parameter and parameter value.

Default Location

x:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\ODS\Xp_param

Running the Sample

To create the extended stored procedure DLLs for these samples, in the appropriate sample subdirectory, open the supplied .dsw file in Microsoft Visual C++.

To run this program

- 1. Build the Xp_param.dll.
- 2. Place the compiled Xp_param.dll in the Microsoft SQL Server\80\Tools\Binn directory.
- 3. Start the server.
- 4. Start SQL Query Analyzer, and then run the Xp_param.sql script.

Remarks

See Also

Samples

Using xp_gettable_odbc

This Microsoft® Visual C++® sample shows using ODBC to open a bound connection to the same instance of Microsoft SQL Server[™] that called the extended stored procedure. The server returns a result set to the extended stored procedure, which passes the result set to the client.

You must create an ODBC datasource called "local" or edit the code to change the datasource name.

Default Location

```
x:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\ODS\Xp_odbc
```

Running the Sample

To create the extended stored procedure DLLs for these samples, go to the appropriate sample subdirectory and open the supplied .dsw file with Microsoft Visual C++.

To run this program

- 1. Build and compile the Xp_odbc.dll.
- 2. Place the compiled Xp_odbc.dll in the Microsoft SQL Server\80\Tools\Binn directory.
- 3. Start the server.
- 4. Start SQL Query Analyzer, and then run the Xp_odbc.sql script.

Remarks

xp_gettable_odbc works only on an instance of SQL Server running on Microsoft Windows® 2000 or Microsoft Windows NT® 4.0.

See Also

Using Bound Connections

Samples

Using xp_gettable_dblib

This Microsoft[®] Visual C++[®] sample shows using the DB-Library interface to open a bound connection to the same instance of Microsoft SQL Server[™] that called the extended stored procedure. The server returns a result set to the extended stored procedure, which passes the result set to the client.

Default Location

x:\Program Files\Microsoft SQL
Server\80\Tools\Devtools\Samples\ODS\Xp_dblib

Running the Sample

To create the extended stored procedure DLLs for these samples, go to the appropriate sample subdirectory and open the supplied .dsw file with Microsoft Visual C++.

To run this program

- 1. Build the Xp_dblib.dll.
- 2. Place the compiled Xp_dblib.dll in the Microsoft SQL Server\80\Tools\Binn directory.
- 3. Start the server.
- 4. Start SQL Query Analyzer, and then run the Xp_dblib.sql script.

See Also

Using Bound Connections

Samples