User Guide

RPG2SQL Integrator

Document Version 1.50.5

RJS Software Systems
2970 Judicial Road, Suite 100
Burnsville, MN 55337

Phone: (952) 736-5800
Fax: (952) 736-5801

Sales e-mail: sales@rjssoftware.com
Support e-mail: support@rjssoftware.com
Web site: http://www.rjssoftware.com

Printed: March 2011

Table of Contents

RPG2SQL Integrator

RPG2SQL Integrator

Welcome to the *RPG2SQL User Guide*. This guide is document version 1.50.5.

Overview

Overview

Introduction to RPG2SQL Integrator

The iSeries RPG2SQL Integrator is a **native OS/400 API** product used to allow ILE/RPG, ILE/Cobol and programmers to directly access any network attached database such as **Oracle**, **Microsoft SQL Server**, **Access, dBase, Foxpro, Excel, CSV, MySQL** and more right from an ILE/RPG or ILE/Cobol program.

Note: While RPG2SQL Integrator works with products like Oracle, Microsoft SQL Server, Access, dBase, Foxpro, Excel, CSV, MySQL and more, RPG2SQL Integrator does not provide the ODBC drivers or OLE DB providers for these products. Contact the database vendors directly for ODBC drivers or OLE DB providers.

Since the RPG2SQL API uses **ADO (Microsoft Active Data Objects)** for database connectivity, any database type that can be opened via ADO or ODBC can be accessed directly from an RPG program.

Prior to the RPG2SQL integrator, AS/400 RPG developers had to resort to manually uploading and downloading data files between the AS/400 and other platforms, implementing file replication software or other means of sharing data between the AS/400 and other databases.

With the RPG2SQL Integrator, those days are over because the RPG2SQL API allows RPG and Cobol programs to directly read, write, update and delete records from remote SQL databases.

- Imagine no more file replication for an AS/400 application to access information in a SQL Server database.
- Imagine directly reading any record in a SQL Server database.
- Imagine updating information into a SQL database right from an AS/400 program.
- Imaging performing a direct daily update to a network based spreadsheet file right from an AS/400 program.
- Imagine creating a Access database for a salesperson to receive their daily sales information.

Integration between the AS/400 and other databases has never been so easy.

The uses for the iSeries RPG2SQL API as a SQL database integrator are endless.

Native Ability to generate Microsoft Excel worksheets.

Make sure to tell us how you're going to use our software. Your ideas are what keep our products growing.

About RJS Software Systems, Inc.

RJS Software is a privately-held software and hardware company dedicated to providing high-quality AS/400 - iSeries, Client/Server and web-based products and customer services. Customer Service is central to the company's objective. Read more about us on our web site.

Licensing Information

iSeries RPG2SQL Integrator is licensed on a per AS/400 system partition basis.

The software can also be loaded on up to 2 network servers without incurring additional License charges.

Contacting RJS Software Systems, Inc.

For technical support, please review the following information before contacting RJS Software Systems.

For Technical Support with iSeries RPG2SQL Integrator
Because of the complexity of dealing with the various AS/400 connectivity environments, please gather and organize as much information as possible on the problem prior to contacting RJS Software Systems for support.

If you have a question about an iSeries RPG2SQL Integrator operation, first browse through the Online Help to try to find your answer prior to calling RJS Software Systems.

Contact Information
Telephone support is available on normal business days from 9:00 am to 5:00 pm central time.
(952) 736-5800 Voice
(952) 736-5801 Fax

Support is available via Email at **support@rjssoftware.com.**

You may also find the answer to your question on our web site: www.rjssoftware.com.

Installation

Installation

Introduction to Installation

This section covers the installation of the iSeries RPG2SQL Integrator software on the AS/400 - iSeries system and PC server.

AS/400 Prerequisites

- A RISC AS/400 system running **V4R2-V5R2** or above is required to run the software.
- TCP/IP connectivity must be enabled. If TCP/IP is not active, the software will not work.

PC Prerequisites

- At least one PC or network server running Windows NT/2000/XP is required to run the network server component for the iSeries RPG2SQL Integrator.

Note: The **iSeries RPG2SQL Integrator Server** PC component can be installed on any PC running **Windows 95, 98, Me, NT, 2000, XP,** however Windows NT/2000/XP is recommended for its stability and performance.

- deally the PC running the iSeries RPG2SQL server PC component should be as fast as possible and should also have a high-speed (100 megabit or above) network connection available to maximize performance.

Note: A high speed server PC is not required, however a high speed PC will help improve overall SQL database access performance.

- **TCP/IP** must be running on the PC and the **iSeries RPG2SQL Integrator Server** component must also be installed and running on the PC. The RPG2SQL Integrator PC server uses **TCP/IP port 22001**.

Client Access/400 IS NOT required.

- The Microsoft Data Access Components contain all the current ADO/ODBC drivers for use by the RPG2SQL Integrator PC server software.

This software only needs to be installed on the RPG2SQL Integrator Server PC.

Ideally Version 2.6 or above of the MDAC components should be installed on the Server PC.

If you're not sure what version of the MDAC components you have loaded, download MDAC version 2.6 or above from the the listed Microsoft or RJS web sites.

Microsoft Data Access Components web link

http://www.microsoft.com/data

RJS Microsoft Data Access Components V2.6 web link
http://www.rjssoftware.com/FILES/mdac_typ.exe

After downloading, run the MDAC installation process and follow all defaults.

Note: The MDAC controls are Microsoft components and not written or packaged by RJS. Any issues caused by the installation of MDAC updates must be addressed by Microsoft.

AS/400 Installation

This section includes instructions for restoring the iSeries RPG2SQL Integrator AS/400 library.

AS/400 Installation Using RSTLIB command
Insert the RJS CD into the AS/400 CD Drive and sign on to the AS/400 with a user ID that has security officer authority.

Run the following AS/400 command to restore the iSeries RPG2SQL Integrator library:
RSTLIB SAVLIB(RJSRPGSQL) DEV(OPT01) OPTFILE('/RJSRPGSQ')

AS/400 Installation Using FTP
This method of installation is described in the RJS Product Catalog and also on the RJS Software Systems installation screen on the RJS CD-ROM.

Note: All web download packages contain an automated FTP upload utility.

AS/400 User Authorization Code Entry

Sign on to the AS/400 with a user ID that has security officer authority.

Entering iSeries RPG2SQL Integrator Access Code
Add the RJSRPGSQL library to the user library list by typing: **ADDLIBLE RJSRPGSQL** and pressing Enter

Go to the iSeries RPG2SQL Integrator main menu by typing: **GO RJSRPGSQL** and pressing Enter.

Take **option 1** and enter the appropriate access codes listed on the access code sheet provided by RJS Software Systems.

If you have questions or problems with entering the access codes, contact RJS Software Systems for assistance.

Installing iSeries RPG2SQL Integrator PC component

The following steps should be used to install iSeries RPG2SQL Integrator on a PC.

Start your PC and make sure Windows is running.

Insert the RJS installation CD-ROM into drive D: (CD-DRIVE).

1.) Click the Windows Start button and select the Run menu. When the Run dialog is shown, type **D:\RPGSQLSV\SETUP.EXE** and click the OK button to start the RJS iSeries RPG2SQL Integrator installation process. When the installation program has started, follow the on screen instructions to complete the PC installation.

Most of the time you should be able to simply use the default settings.

After copying files, the Setup program will create a **RJS iSeries RPG2SQL Integrator** program group and notify you when it's done.

The **RJS iSeries RPG2SQL Integrator** can now be launched from the **RJS iSeries RPG2SQL Integrator** program group.

Deinstallation of iSeries RPGSQL Integrator

The following sections describe how to remove the iSeries RPG2SQL Integrator from an AS/400 system.

Removing iSeries RPG2SQL Integrator from an AS/400
To remove iSeries RPG2SQL Integrator from an AS/400 system, perform the following individual steps.

Make sure the RJSRPGSQL library is not being used.

Delete the RJSRPGSQL library from the AS/400 by entering the following command:
DLTLIB LIB(RJSRPGSQL)

Removing iSeries RPG2SQL Integrator from a PC
Remove the **C:\Program Files\RPGSQLSV** directory from the PC.

Getting Started

Getting Started

Introduction to Getting Started

The purpose of this section is to describe how to quickly get started using the iSeries RPG2SQL Integrator to start developing RPG and COBOL programs that can access SQL Server and other ADO (Microsoft Active Data Objects) or ODBC databases.

Installing the iSeries RPG2SQL Integrator Software

If you haven't installed the software yet, go back to the installation section and install the software before continuing. If you're just reading ahead to find out what to do, then please continue.

Overview of RPGSQL Integrator API

The iSeries RPG2SQL Integrator is a **native OS/400 API** product used to allow ILE/RPG, ILE/Cobol and programmers to directly access any network attached database such as **Oracle**, **Microsoft SQL Server**, **Access, dBase, Foxpro, Excel, CSV, MySQL** and more right from an ILE/RPG or ILE/Cobol program.

Note: While RPG2SQL Integrator works with products like Oracle, Microsoft SQL Server, Access, dBase, Foxpro, Excel, CSV, MySQL and more, RPG2SQL Integrator does not provide the ODBC drivers or OLE DB providers for these products. Contact the database vendors directly for ODBC drivers or OLE DB providers.

Since the RPG2SQL API uses **ADO (Microsoft Active Data Objects)** for database connectivity, any database type that can be opened via ADO or ODBC can be accessed directly from an RPG program.

Prior to the RPG2SQL Integrator, AS/400 RPG developers has to resort to manually uploading and downloading data files between the AS/400 and other platforms, implementing file replication software or other means of sharing data between the AS/400 and other databases.

With the RPG2SQL Integrator, those days are over because the RPG2SQL API allows RPG and Cobol programs to directly read, write, update and delete records from remote SQL databases.

- Imagine no more file replication for an AS/400 application to access information in a SQL Server database.
- Imagine directly reading any record in a SQL Server database.
- Imagine updating information into a SQL database right from an AS/400 program.
- Imaging performing a direct daily update to a network based spreadsheet file right from an AS/400 program.
- Imagine creating a Access database for a salesperson to receive their daily sales information.

Integration between the AS/400 and other databases has never been so easy.

The uses for the iSeries RPG2SQL API as an SQL database integrator are endless.

Make sure to tell us how you're going to use our software. Your ideas are what keep our products growing.

ADO/ODBC Data Source Examples

The RPG2SQL Integrator uses ADO or ODBC data sources on the PC to talk to the various PC databases. Listed below are some example strings for passing to the **SQL_DBOpenConn** function. More example strings can be found in member **ADODSNSAMP** in **RJSRPGSQL/SOURCE**.

```
 *--------------------------------------------------------------------
 * Use a Predefined ODBC Data Source (DSN)
 *--------------------------------------------------------------------
C                   Eval      DataSrc='DSN=myDSN;' +
C                             'Uid=myUser;' +
C                             'Pwd=myPassword;'
```

The example listed above opens an existing ODBC data source named **myDSN** defined via the Windows **ODBC Administrator** program. To start the ODBC Administrator program, select **Start/Run**, type **ODBCAD32.EXE** and click OK.

```
 *--------------------------------------------------------------------
 * Microsoft Access Table (DSNless Connection String)
 *--------------------------------------------------------------------
C                   Eval      DataSrc='Driver={Microsoft Access Driver
C                             '(*.mdb)}; ' +
C                             'Dbq=c:\program files\' +
C                             'rpgsqlsv\irpgsql.mdb; ' +
C                             'Uid=admin;' +
C                             'Pwd=;'
```

The example listed above opens an Access database without the need to define an ODBC data source via the Windows ODDBC Administrator program. This is known as a DSNless ADO connection string.

```
 *--------------------------------------------------------------------
 * Microsoft Excel (DSNless Connection String)
 *--------------------------------------------------------------------
C                   Eval      DataSrc='Driver={Microsoft Excel Driver '
C                             '(*.xls)}; ' +
C                             'DriverID=790; ' +
C                             'Dbq=c:\temp\test.xls; ' +
C                             'DefaultDir=c:\temp;' +
C                             'Pwd=;'
```

The example listed above opens an Excel workbook without the need to define an ODBC data source via the Windows ODDBC Administrator program. This is known as a DSNless ADO connection string.

```
 *--------------------------------------------------------------------
 * Microsoft Dbase Driver (DSNless Connection String)
 *--------------------------------------------------------------------
C                   Eval      DataSrc='Driver={Microsoft dBASE Driver '
C                             '(*.dbf)}; ' +
C                             'DriverID=277; ' +
C                             'Dbq=c:\temp; '
```

The example listed above sets a dBase work directory without the need to define an ODBC data source via the Windows ODDBC Administrator program. This is known as a DSNless ADO connection string. When selecting records from a table in this directory, the following SQL example would be used:

select * from NameAddress.dbf

```
 *--------------------------------------------------------------------
 * Microsoft SQL Server (DSNless Connection String)
 *--------------------------------------------------------------------
C                   Eval      DataSrc='Driver={SQL Server}; ' +
C                             'Server=1.1.1.1;' +
C                             'Database=pubs;' +
C                             'Uid=sa;' +
C                             'Pwd=;'
```

The example listed above connects to a SQL Server database without the need to define an ODBC data source via the Windows ODDBC Administrator program. This is known as a DSNless ADO connection string. SQL Server needs a valid SQL user ID and password passed to it to log in. Check with your SQL Server administrator for a valid user id and password for your SQL Server.

If you're not sure whether you have a particular ADO/ODBC driver available on the PC running the RPG2SQL Integrator, contact RJS for assistance.

How to Use RPG2SQL Integrator Sample Code

How to Use RPG2SQL Integrator Sample Code

RPG2SQL Integrator has several examples to help the developer use this tool kit. Items 12 and up on the **RJSRPGSQL/RJSRPGSQL** main menu have source code in the file **RJSRPGSQL/SOURCE**. There are many more examples in that source file than are on the menu.
The RPG2SQL Integrator product consists of the following service programs that contain the procedures you can use to work with remote databases:

- **RPGSQL01R** This service program contains all the procedures and can be used with both **ILE RPG** and **ILE COBOL**.
- **RPGDDS01R** This service program contains the procedures for DDS-based data retrieval and can be used with both **ILE RPG** and **ILE COBOL**.
- **RPGSQL02R** (deprecated) This service program was created for use with **ILE COBOL** only – it is no longer needed, is not being updated, and does not contain any new procedures (see the **COBOL** examples **SCTEST***, as well as copy books **RPGSQL1CB** and **RPGSQL2CB** for details on the use of **RPGSQL01R** with **ILE COBOL** programs).

Structure of the Examples

All examples comprise 4 or 5 items:

- A command, such as **SQTEST01**.
- A command processing program, such as **SQTEST01C**.
- An **RPGLE** program that has the real sample code, such as **SQTEST01R**.
- A program to build the example, such as **SQTEST01B** (no longer needed, since a header specification on the **RPGLE** source lets you use PDM option 14 to create it).
- Some examples might include a printer file or display file.

All examples have a command front end. The reason for this structure is to simplify running the examples and is not intended to show typical usage.

Locating Source for Menu Items

In order to tell which example corresponds to a menu item, simply type the item number and press **Enter**. The sample command name is part of the header text of the command prompt. For example, option 14 brings up the following:

```
Read from Excel (SQTEST04)
```

All the source members for this example will start with **SQTEST04**

Required Items for Creating an RPG2SQL Application

When using RPG2SQL Integrator you must:

1. Be sure **RJSRPGSQL** is in your library list when creating the program or service program.
2. Include an H-specification that includes the **RJSRPGSQL** binding directory, as well as QC2LE, the binding directory for the C runtime library functions, as follows:

```
BNDDIR('RJSRPGSQL':'QC2LE')
```

3. Include the **RPG2SQL Integrator** copy member at the beginning of your D-specifications, as follows.

```
/COPY SOURCE,RPGSQLH
```

4. Convert the IP address of the RPG2SQL Integrator PC component to its dotted IP form, as follows:

```
C                   Eval      Rtn = TCP_GetIPFromHost(IPAddr : DotIPAddr)
```

5. 5. Set the multiple buffer flag just before executing SQL_Connect to ensure that all data is returned to your program, as follows:

```
C                   CallP     SQL_TCPMultBuff(1)
```

Basic Structure of an RPG2SQL Integrator Application

Processing steps:

- Connect to the RPG2SQL Integrator PC component (**SQL_Connect**).
- Connect to the remote database (**SQL_DBOpenConn**).
- Do the work, such as reading iSeries tables and sending data to the remote database.
- Disconnect from the remote database (**SQL_DBCloseConn**).
- Disconnect from the RPG2SQL Integrator PC component (**SQL_Disconnect**).

RPG2SQL Integrator Copy Member

The copy member for all **RPG2SQL Integrator** applications is member **RPGSQLH** in file **RJSRPGSQL/SOURCE**. This copy member contains:

- Prototypes for all the procedures in **RPG2SQL Integrator**.
- Several data “types” that you can use in your programs, including **TChar**, for a 1-character variable, or **TString**, for a 256-character variable, or **Tint**, for a 4-byte integer (10I 0).
- Several stored procedure and field type constants.
- Several Excel-specific constants.

Error Handling

Error handling is kept simple in all the examples, in order to keep the examples clean. For a basic example of error handling, see source member **RSQCONNR**, which shows how to use the **SQL_LastFullErr** procedure.

Program Flow when Reading Selected Records from SQL table

The following pseudocode shows how easy it is to write an RPG program to read all records from any PC database file.

SQL_Connect - Connect to RPG2SQL PC server component

SQL_DBOpenConn - Open connection to selected ADO data source

SQL_RunSQLSel - Run a SQL record selection statement to create a recordset containing the selected records.

SQL_MoveFirstBuf - Move to first record in file.

Begin of Loop (Do until EOF) - Loop through all records and retrieve all required fields from current record.

'Extract fields from record using the following:
Use SQL_GetFldCharB, SQL_GetFldNumB or SQL_GetFldDatB function calls to retrieve selected field values.

'Write extracted fields to AS/400 table
Write Record

SQL_MoveNextBuf - Move to next record in file.

End of Loop

SQL_DBCloseConn - Close connection to selected ADO data source

SQL_Disconnect - Disconnect from RPG2SQL PC server component

Program Flow when inserting new records into SQL table

The following pseudocode shows how easy it is to write an RPG program to write all records from any AS/400 file to a PC database file.

SQL_Connect - Connect to RPG2SQL PC server component

SQL_DBOpenConn - Open connection to selected ADO data source

Read AS400FILE - Read first record from AS/400 data file

'Loop to read entire AS/400 data file
Begin of Loop (Do until EOF)

SQL_RunSQLExec - Build and run a SQL record INSERT statement from AS/400 file fields to create a new record in SQL database.

Read AS400FILE - Read next record from AS/400 data file

End of Loop

SQL_DBCloseConn - Close connection to selected ADO data source

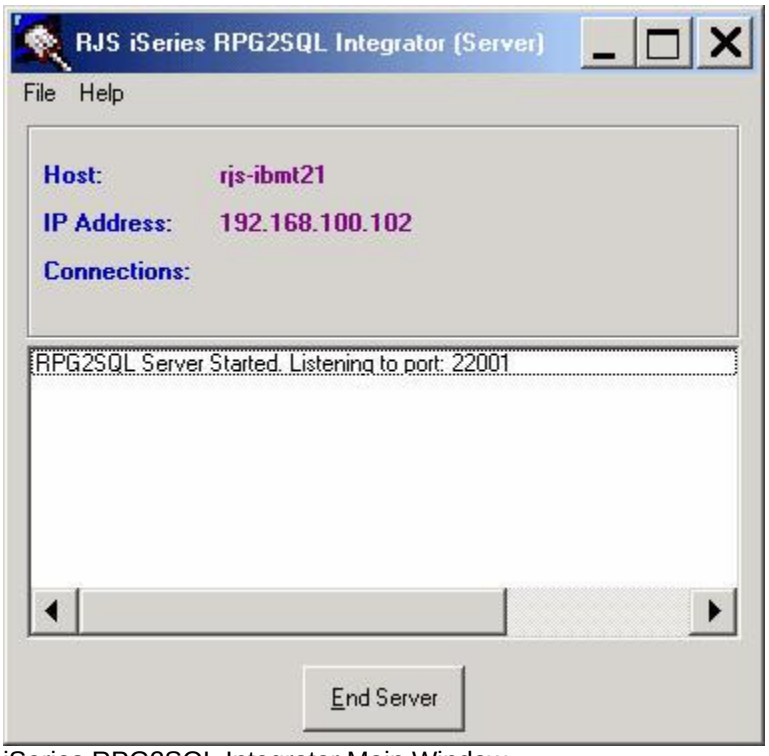
SQL_Disconnect - Disconnect from RPG2SQL PC server component

Starting iSeries RPG2SQL Integrator Server on PC

In order for RPG programs to talk to a PC database, the RPG2SQL Integrator PC server must be started and running on at least one PC.

Note: Make sure the RJS iSeries RPG2SQL Integrator has been installed on the PC before continuing.

Start the **RJS iSeries RPG2SQL Integrator Server - Debug Mode** program from the **RJS iSeries RPG2SQL Integrator** program group.



iSeries RPG2SQL Integrator Main Window

The server automatically starts listening on TCP/IP **port 22001**, so you can simply minimize the main window and the program will hide in the system tray.

The PC is now ready to receive SQL database requests.

The iSeries RPG2SQL Integrator program only needs to be started once. It will remain running until the PC is shut down or the program is closed.

Note: Only run the program in **Debug Mode** during initial testing or troubleshooting. Debug Mode greatly impacts performance.

Logging Activity
To capture all activity in a log file, go to **File/Settings** and check the **Enable Logging** check box. Then you must exit RPG2SQL Integrator and restart it.

Note: You can limit the size of the log file using the **Maximum Log File Size (Bytes)** text box.

Sending Debug info to RJS for Analysis
If you're having problems with a particular SQL query, you can use the **File/Export Debug Info to Clipboard** menu option to send your debug log to RJS for analysis.

Ending the RPG2SQL Integrator PC Server program
To end the program, simply bring up the main window and click the **End** button.

First Steps - Running Sample Programs

Each of the included program samples are compiled and ready to run from the main menu.

Try out these samples to quickly see how the iSeries RPG2SQL Integrator works.

Note: Make sure the RPG2SQL Integrator PC Server is running before attempting to run any of these menu options. Otherwise the samples will not work correctly.

Type **GO RJSRPGSQL** and press Enter to get to the main menu.

Listed below are the sample program menu options and a description of what will happen when the option is run:

Option 10 - Work with Sample Source Code
This menu option is used to display all the included sample programs in the **SOURCE** source file in the **RJSRPGSQL** library.

Note: This option assumes that you have the IBM Application Development software tools on your AS/400 system and are familiar with using them.

Option 11 - Test ADO/ODBC Database Connection - (SQLTEST02C)
This menu option is used to see if the RPG2SQL Integrator PC Server is running.

Option 12 - Write records to Access database IRPGSQL.MDB
This menu option deletes all existing records in the **NameAddress** MS Access table in the **C:\Program Files\RPGSQLSV\IRPGSQL.MDB** access database and then inserts ten duplicate records.

Option 13 - Read all records from Access Database IRPGSQL.MDB

This menu option reads all records from the NameAddress Access Table in file **C:\Program Files\RPGSQLSV\IRPGSQL.MDB**

Option 14 - Read all records from Sample Excel Workbook - NameAddr.XLS
This menu option reads all records from the NameAddress workbook in file **C:\Program Files\RPGSQLSV\NAMEADDR.XLS**

Option 15 - Read all records from Sample dBase Table - NameAddr.DBF
This menu option reads all records from the NameAddr dBase table in file **C:\Program Files\RPGSQLSV\NAMEADDR.DBF**

Option 16 - Read all records from Sample CSV Table - NameAddr.CSV
This menu option reads all records from the NameAddr CSV table in file **C:\Program Files\RPGSQLSV\NAMEADDR.CSV**

Option 20 - Create SQL Server Table NameAddr and Insert Records
This menu option creates a table named **NameAddr** in the **pubs** database on the selected SQL Server database server and inserts records from the AS/400 file: **CUSTOMER**.

Option 21 - Illustrate use of Commit/Rollback SQL Transaction
This menu option creates a table named **NameAddr** in the **pubs** database on the selected SQL Server database server and inserts records from the AS/400 file: **CUSTOMER**.

Note: The difference between this option and option 20 is that this sample program illustrates the use of committment control and SQL Server transactions.

Option 22 - Read All Records from NameAddr Table and Display
This menu option reads all records from the **NameAddr** table in the **pubs** database on the selected SQL Server database server, writes the data to a temporary AS/400 file and displays all the record data.

What's Next After Getting Started?

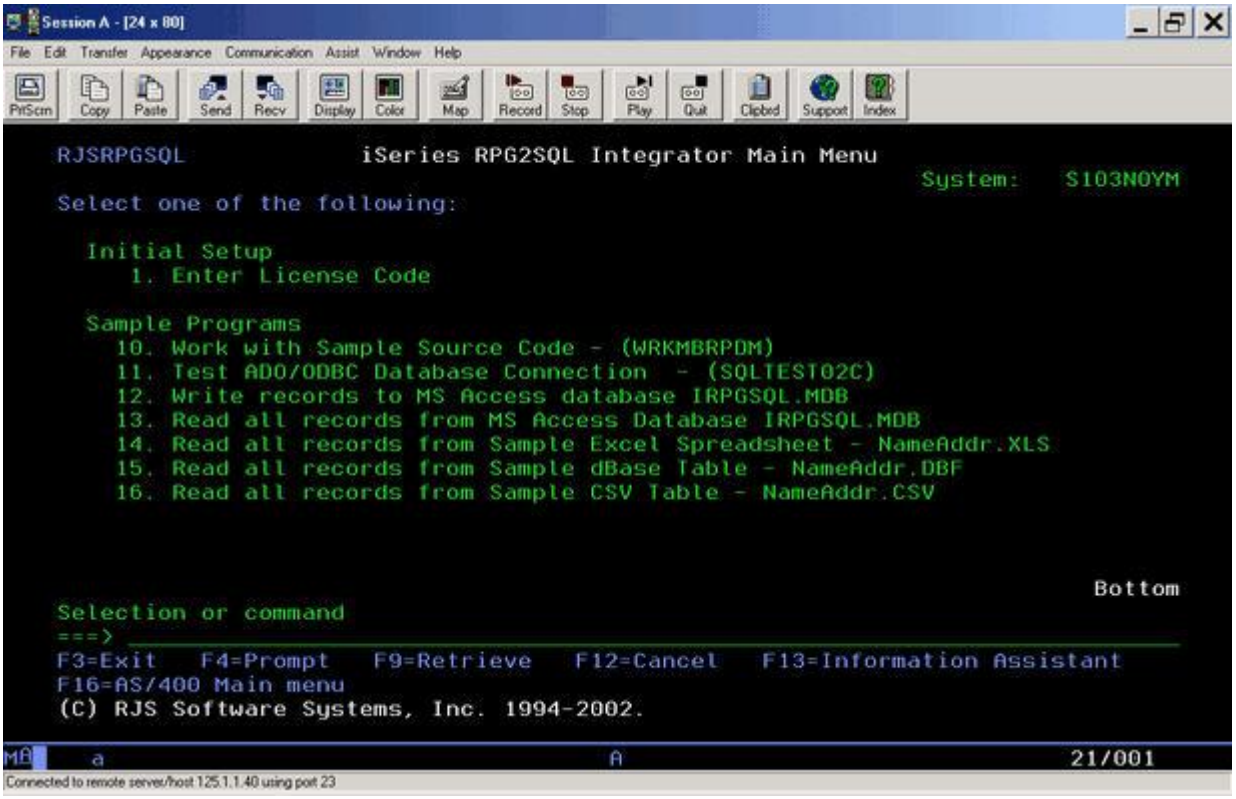
You should now have the basics down for using the RPG2SQL Integrator API. The next few sections are a reference for each of the iSeries RPG2SQL Integrator commands and available parameters.

Main Menu

Main Menu

Introduction to Main Menu

This section gives an overview of the iSeries RPG2SQL Integrator main menu and the available options.



iSeries RPG2SQL Integrator Main Menu

Getting to the Main Menu

Sign on to the AS/400 system and add the RJSRPGSQL library to your library list using the following AS/400 command: **ADDLIBLE RJSRPGSQL**

Next type: **GO RJSRPGSQL** and press Enter. This will take you to the iSeries RPG2SQL Integrator main menu.

Available Options on the Main Menu

Option 1 - Enter iSeries RPG2SQL Integrator License Code
This menu option is used to enter the AS/400 access code for the iSeries RPG2SQL Integrator software. An access code will be provided to you by RJS Software Systems Inc.

Option 10 - Work with Sample Source Code
This menu option is used to display all the included sample programs in the **SOURCE** source file in the **RJSRPGSQL** library.

Option 11 - Test ADO/ODBC Database Connection - (SQLTEST02C)
This menu option is used to see if the RPG2SQL Integrator PC Server is running.

Option 12 - Write records to Access database IRPGSQL.MDB
This menu option deletes all existing records in the **NameAddress** Access table in the **C:\Program Files\RPGSQLSV\IRPGSQL.MDB** access database and then inserts ten duplicate records.

Option 13 - Read all records from Access Database IRPGSQL.MDB
This menu option reads all records from the NameAddress Access Table in file **C:\Program Files\RPGSQLSV\IRPGSQL.MDB**

Option 14 - Read all records from Sample Excel Workbook - NameAddr.XLS
This menu option reads all records from the NameAddress workbook in file **C:\Program Files\RPGSQLSV\NAMEADDR.XLS**

Option 15 - Read all records from Sample dBase Table - NameAddr.DBF
This menu option reads all records from the NameAddr dBase table in file **C:\Program Files\RPGSQLSV\NAMEADDR.DBF**

Option 16 - Read all records from Sample CSV Table - NameAddr.CSV
This menu option reads all records from the NameAddr CSV table in file **C:\Program Files\RPGSQLSV\NAMEADDR.CSV**

Option 20 - Create SQL Server Table NameAddr and Insert Records
This menu option creates a table named **NameAddr** in the **pubs** database on the selected SQL Server database server and inserts records from the AS/400 file: **CUSTOMER**.

Option 21 - Illustrate use of Commit/Rollback SQL Transaction
This menu option creates a table named **NameAddr** in the **pubs** database on the selected SQL Server database server and inserts records from the AS/400 file: **CUSTOMER**.

Note: The difference between this option and option 20 is that this sample program illustrates the use of committment control and SQL Server transactions.

Option 22 - Read All Records from NameAddr Table and Display
This menu option reads all records from the **NameAddr** table in the **pubs** database on the selected SQL Server database server, writes the data to a temporary AS/400 file and displays all the record data.

RPG2SQL ILE RPG Service Program Procedures

RPG2SQL ILE RPG Service Program Procedures

8 SQL_NonBlocking

The **SQL_NonBlocking** API is used to set socket connections to nonblocking. You can use this API to eliminate time waits on the iSeries.

Turning on nonblocking causes connections to the RPG2SQL Integrator server to timeout after a specified number of seconds when the socket is unusable for any reason. Functions that return a numeric value will return unknown communications failure (-3). Functions that return a buffer will return *ERROR*.

Note: The **SQL_NonBlocking** API must be called before the **SQL_Connect** API if nonblocking is desired.

Note: Blocking connections are the default.

Note: You will *not* be able to use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why an API call failed due to timeout.

Returns
0 = Blocking/nonblocking was set successfully.

-2 = Unknown error.

RPG Prototype
```
DSQL_NonBlocking  Pr                  Like(TInt)
D OnOff                               Like(TInt) Value
D Timeout                             Like(TInt) Value
```

Parameters
1.) OnOff - Use nonblocking socket connections. 0=blocking. 1=nonblocking with timeout.

2.) Timeout - Set nonblocking timeout in seconds. If set to 0, timeout defaults to 60 seconds.

Example Usage
```
 *------------------------------------------------------------------
 *                  ** Set connection to nonblocking
 *                  ** with 30 second timeout.
 *------------------------------------------------------------------
C                   CallP     SQL_NonBlocking(1:30)
```

TCP_GetIPFromHost

The **TCP_GetIPFromHost** API is used to convert a fully qualified domain name into its dotted IP address. All other functions that use an IP address use the dotted IP format.

Note: A dotted IP address will be left as is.

Returns
0 = Domain name was converted successfully.

-11 = Domain name could not be found.

-12 = 4-byte IP address could not be converted to dotted IP format.

RPG Prototype
```
DTCP_GetIPFromHost...
D                 Pr                  like(TInt)
D IP_Addr                        255  Value
D RtnDotIP                        15
```

Parameters
1.) IP_Addr - Fully qualified domain name.

2.) RtnDotIP - Dotted IP address.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Convert youri.company.com to
 *                  ** dotted IP format.
 *-------------------------------------------------------------
C                   CallP     TCP_GetIPFromHost('youri.company.com':
C                                               RtnDotIP)
```

SQL_TCPTrace

The **SQL_TCPTrace** API is used to enable or disable TCP/IP logging to the RPGSQLLOG file.

Note: TCP/IP logging is off by default.

Returns
0 = TCP/IP logging was set successfully.

-2 = Unknown error.

RPG Prototype

```
DSQL_TCPTrace     Pr                  Like(TInt)
D OnOff                               Like(TInt) Value
```

Parameters
1.) OnOff - Set TCP/IP logging. 0=disable logging. 1=enable logging.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Enable TCP/IP logging.
 *-------------------------------------------------------------
C                   CallP     SQL_TCPTrace(1)
```

SQL_TCPMultBuff

The **SQL_TCPMultBuff** API is used to enable or disable TCP/IP multiple buffering. You can use this API to overcome packet truncation issues.

When disabled, the socket will be read 3 times. When enabled, the socket will be read until an end-of-data marker is received.

Note: TCP/IP multiple buffering is off by default. It is strongly recommended that you always enable multiple buffering.

Note: This requires at least V1.0.42 of the RPG2SQL Integrator server software.

Returns
0 = TCP/IP multiple buffering was set successfully.

-2 = Unknown error.

RPG Prototype

```
DSQL_TCPMultBuff  Pr                  Like(TInt)
D OnOff                               Like(TInt) Value
```

Parameters
1.) OnOff - Set TCP/IP multiple buffering. 0=disable multiple buffering. 1=enable multiple buffering.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Enable TCP/IP multiple buffering.
 *-------------------------------------------------------------
C                   CallP     SQL_TCPMultBuff(1)
```

SQL_Connect

The **SQL_Connect** API is the first procedure used in an ILE program that will be accessing SQL data from a remote ODBC/ADO database.

The **SQL_Connect** API connects to the RPG2SQL Integrator PC Server component via TCP/IP sockets. The RPG2SQL Integrator uses TCP/IP port **22001** to communicate with the PC server component.

If the **SQL_Connect** API completes successfully, a socket ID number is returned from the PC server for the current connection. This socket ID is used until a **SQL_Disconnect** is called to end the connection with the PC server.

Returns
TCP/IP Socket ID to use.

-999 = RPG2SQL Server connection was not successful. Potentially the PC server component may not be running or the PC server may have a firewall protecting port 22001.

RPG Prototype

```
DSQL_Connect      Pr                  Like(TInt)
D IP_Addr                             Like(TDotIP) Value
```

Parameters
1.) IP_Addr - TCP/IP Address for RPG2SQL Server PC component.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Connect to RPG SQL Server
 *-------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
C*                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif
```

SQL_Disconnect

The **SQL_Disconnect** API is the last procedure used in an ILE program that will be access SQL data from a remote database. The API disconnects the current AS/400 job from the RPG2SQL Integrator PC Server component, thus freeing up the socket connection for use by another program.

The disconnect should always be called before a program is ended.

Note: If the SQL_Disconnect procedure is not run, a connection to the PC Server will be maintained until the AS/400 job which was accessing the PC server has ended. While this is not usually problematic in nature, it can be if the SQL_Connect function is called several times throughout an RPG program and the connections are not closed after each connection has completed.

Returns
0 = Completed successfully

-2 = Unknown error.

RPG Prototype
```
DSQL_Disconnect  Pr                  Like(Tint)
D Socket                             Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage
```
 *-------------------------------------------------------------
 *                  ** Disconnect from RPG2SQL server
 *-------------------------------------------------------------
C                   callp     SQL_Disconnect(SQL_Socket)
```

SQL_Timeout

The **SQL_Timeout** API is used to set ADO query timeouts.

Note: This should always be run before the **SQL_DBOpenConn** or **SQL_RunSQLSel** API.

Returns
0 = Unlimited seconds query timeout.

Set number of seconds for query timeout.

-1 = ADO connection error.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype
```
DSQL_TimeOut      Pr                 Like(TInt)
D Socket                             Like(TInt) Value
D Timeout                            Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Timeout - Set ADO query timeout in seconds.

Note: Not all databases require all parameters.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Set SQL query timeout for 90 seconds.
 *-------------------------------------------------------------
C                   Eval      Rtn = SQL_Timeout(SQL_Socket:90)
```

SQL_DBOpenConn

The **SQL_DBOpenConn** API is the second procedure used in an ILE program that will be access SQL data from a remote database.

The **SQL_DBOpenConn** API connects the ILE program to the specified SQL database using ADO (Microsoft Active Data Obects).

Note: A valid ADO data source must be passed to the SQL_DBOpenConn API call or the database connection will not work correctly.

If the **SQL_DBOpenConn** API completes successfully, a socket ID number is returned from the PC server for the current connection. This socket ID is used until a SQL_Disconnect is called to end the connection with the PC server.

Returns
0 = ADO/SQL database connection completed successfully

-1 = ADO/SQL database connection failed.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

-999 = Invalid license

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_DBOpenConn   Pr                  Like(Tint)
D Socket                              Like(TInt) Value
D DataSource                          Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Data Source - Formatted ADO data source containing driver type, server, database, userid and password for SQL database.

Note: Not all databases require all parameters.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Open ADO SQL Server Database Connection
 *                  ** to the SQL Server Pubs database.
 *-------------------------------------------------------------
C                   Eval      Rtn = SQL_DBOpenConn(SQL_Socket:
C                             'Driver={SQL Server};Server=10.1.1.1'+
C                             ';Database=Pubs;Uid=sa;' +
C                             'Pwd=admin;')
```

SQL_DBCloseConn

The **SQL_DBCloseConn** API closes a currently opened database connection to the specified ADO/SQL database on the RPG2SQL server.

Returns
0 = ADO/SQL database connection completed successfully

-1 = ADO/SQL database connection failed.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_DBCloseConn  Pr                  Like(Tint)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Close ADO Database Conection
 *-------------------------------------------------------------
C                   callp     SQL_DBCloseConn(SQL_Socket)
```

SQL_BeginTran

The **SQL_BeginTran** API starts an SQL Server database transaction. After issuing a begin transaction, the **SQL_CommitTran** or **SQL_Rollback** procedure must be used to complete a transaction or roll back (remove) changes from the database.

Using database transactions is an optional feature of Microsoft SQL Server and some other database types to buffer changes made via INSERT, UPDATE and DELETE statements until all changes have been made. If any updates fail, database changes can be removed in a single rollback operation. Once all changes are made, a commit operation is performed to save all changes to the database.

Note: There is a lot of detail in the Microsoft SQL Server Online Help regarding using database transactions and when they should be used.

Note: Using transactions is not required and only works with Microsoft SQL Server or other ADO/ODBC databases that support database transaction control.

The available RPG2SQL Integrator transaction processing functions are:
SQL_BeginTran - Start a database transaction.
SQL_CommitTran - Commit all changes to the selected database.
SQL_Rollback - Rollback changes made during a database transaction.

All INSERT, UPDATE and DELETE statements that are run after the **SQL_BeginTran** procedure will be treated as being part of the same database transaction. Once you have run all selected SQL statements, make sure to issue the **SQL_CommitTran** to commit all changes to the SQL Server database.

Note: If **SQL_CommitTran** is not run, all changes made to the database since running **SQL_BeginTran** can potentially be lost.

If a transaction is started with **SQL_BeginTran**, all INSERT, UPDATE or DELETE statements can be rolled back prior to ending the transaction by using a single call to the **SQL_Rollback** procedure. Once a rollback has been called, all changes made after the **SQL_BeginTran** are removed from the SQL Server database tables.

After issuing a **SQL_CommitTran** or **SQL_Rollback** call, you will need to issue the **SQL_BeginTran** call to start a new SQL Server database transaction.

Returns
0 = ADO/SQL begin transaction completed successfully

-1 = ADO/SQL begin transaction failed.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype
```
DSQL_BeginTran    Pr                  Like(Tint)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage
```
 *------------------------------------------------------------------
 *                  ** Start SQL Server database transaction
 *------------------------------------------------------------------
C                   Eval     Rtn=SQL_BeginTran(SQL_Socket)
```

SQL_CommitTran

The **SQL_CommitTran** API permanently saves all changes made during a database transaction. After issuing a commit command to save changes, the **SQL_BeginTran** procedure must be used to start a new database transaction if required.

Using database transactions is an optional feature of Microsoft SQL Server and some other database types to buffer changes made via INSERT, UPDATE and DELETE statements until all changes have been made. If any updates fail, database changes can be removed in a single rollback operation. Once all changes are made, a commit operation is performed to save all changes to the database.

Note: There is a lot of detail in the Microsoft SQL Server Online Help regarding using database transactions and when they should be used.

Note: Using transactions is not required and only works with Microsoft SQL Server or other ADO/ODBC databases that support database transaction control.

The available RPG2SQL Integrator transaction processing functions are:
SQL_BeginTran - Start a database transaction.
SQL_CommitTran - Commit all changes to the selected database.
SQL_Rollback - Rollback changes made during a database transaction.

All INSERT, UPDATE and DELETE statements that are run after the **SQL_BeginTran** procedure will be treated as being part of the same database transaction. Once you have run all selected SQL statements, make sure to issue the **SQL_CommitTran** to commit all changes to the SQL Server database.

Note: If **SQL_CommitTran** is not run, all changes made to the database since running **SQL_BeginTran** can potentially be lost.

If a transaction is started with **SQL_BeginTran**, all INSERT, UPDATE or DELETE statements can be rolled back prior to ending the transaction by using a single call to the **SQL_Rollback** procedure. Once a rollback has been called, all changes made after the **SQL_BeginTran** are removed from the SQL Server database tables.

After issuing a **SQL_CommitTran** or **SQL_Rollback** call, you will need to issue the **SQL_BeginTran** call to start a new SQL Server database transaction.

Returns
0 = ADO/SQL commit transaction completed successfully

-1 = ADO/SQL commit transaction failed.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_CommitTran   Pr                  Like(Tint)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *------------------------------------------------------------
 *                  ** Commit database changes made during
 *                     SQL Server database transaction.
 *------------------------------------------------------------
C                   Eval     Rtn=SQL_CommitTran(SQL_Socket)
```

SQL_Rollback

The **SQL_Rollback** API rolls back (removes) all changes made since **SQL_BeginTran** was used to start a database transaction. After issuing a rollback command, the **SQL_BeginTran** procedure must be used to start a new database transaction if required.

Using database transactions is an optional feature of Microsoft SQL Server and some other database types to buffer changes made via INSERT, UPDATE and DELETE statements until all changes have been made. If any updates fail, database changes can be removed in a single rollback operation. Once all changes are made, a commit operation is performed to save all changes to the database.

Note: There is a lot of detail in the Microsoft SQL Server Online Help regarding using database transactions and when they should be used.

Note: Using transactions is not required and only works with Microsoft SQL Server or other ADO/ODBC databases that support database transaction control.

The available RPG2SQL Integrator transaction processing functions are:
SQL_BeginTran - Start a database transaction.
SQL_CommitTran - Commit all changes to the selected database.
SQL_Rollback - Rollback changes made during a database transaction.

All INSERT, UPDATE and DELETE statements that are run after the **SQL_BeginTran** procedure will be treated as being part of the same database transaction. Once you have run all selected SQL statements, make sure to issue the **SQL_CommitTran** to commit all changes to the SQL Server database.

Note: If **SQL_CommitTran** is not run, all changes made to the database since running **SQL_BeginTran** can potentially be lost.

If a transaction is started with **SQL_BeginTran**, all INSERT, UPDATE or DELETE statements can be rolled back prior to ending the transaction by using a single call to the **SQL_Rollback** procedure. Once a rollback has been called, all changes made after the **SQL_BeginTran** are removed from the SQL Server database tables.

After issuing a **SQL_CommitTran** or **SQL_Rollback** call, you will need to issue the **SQL_BeginTran** call to start a new SQL Server database transaction.

Returns
0 = ADO/SQL commit transaction completed successfully

-1 = ADO/SQL commit transaction failed.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_Rollback     Pr                  Like(Tint)
D Socket                              Like(TInt) Value
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *------------------------------------------------------------------
 *                  ** Rollback (remove) database changes made during
 *                     SQL Server database transaction.
 *------------------------------------------------------------------
C                   Eval     Rtn=SQL_Rollback(SQL_Socket)
```

SQL_RunSQLSel

The **SQL_RunSQLSel** API is used to run a SQL record selection query and open an ADO recordset for reading records from a ADO/SQL database.

Returns

0 = ADO/SQL database operation completed successfully

-1 = ADO/SQL database operation failed.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_RunSQLSel    Pr                  Like(Tint)
D Socket                              Like(TInt) Value
D SQLString                 4096A     Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) SQLString - SQL record selection query string.

Example to select all records from NameAddress table:

```
SELECT * FROM NameAddress
```

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Run SQL Query to Open ADO Recordset
 *-------------------------------------------------------------
C                    Eval      Rtn = SQL_RunSQLSel(SQL_Socket:
C                              'SELECT * FROM NameAddress')
```

SQL_RunSQLExec

The **SQL_RunSQLExec** API is used to run SQL record **INSERT**, **UPDATE** and **DELETE** operations to add data to a n ADO/ODBC SQL table or modify existing records in a table

Returns
0 = ADO/SQL database operation completed successfully

-**1** = ADO/SQL database operation failed.

-2 = Unknown error occurred.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_RunSQLExec  Pr                  Like(Tint)
D Socket                             Like(TInt) Value
D SQLString                  4096A   Value
D                                    Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) SQLString - SQL Insert, Update or Delete string.

Example SQL to insert record:
INSERT into NameAddress (First,Last) VALUES('James','Jones')

Example SQL to update Address1 field in record:
UPDATE NameAddress SET Address1 = '222 Main St' where Last = 'Jones' and First = 'James'

Example SQL to delete record:
DELETE from NameAddress where Last = 'Jones' and First = 'James'

Example SQL to delete all records:
DELETE from NameAddress

Example Usage in RPG program

```
 *------------------------------------------------------------------
 *                  ** Run Insert SQL Query to Insert Record
 *------------------------------------------------------------------
C                    Eval      Rtn = SQL_RunSQLExec(SQL_Socket:
C                              'insert into NameAddress' +
C                              '(First,' +
C                              'Last,' +
C                              'Address1,' +
C                              'Address2,' +
C                              'City,' +
 C                              'State,' +
 C                              'Zip,' +
 C                              'Phone,' +
 C                              'Fax,' +
 C                              'Email,' +
 C                              'Date1,' +
 C                              'Dollars) ' +
 C                              'VALUES(' +
 C                              quot + 'James' + quot + ',' +
 C                              quot + 'Johnson' + quot + ','  +
 C                              quot + '111 Main Street' + quot + ',' +
 C                              quot + 'Apt 5' + quot + ',' +
 C                              quot + 'Mpls' + quot + ',' +
 C                              quot + 'MN' + quot + ',' +
 C                              quot + '55555' + quot + ',' +
 C                              quot + '111-111-1111' + quot + ',' +
 C                              quot + '222-222-2222' + quot + ',' +
 C                              quot + 'james@johnson.com' + quot + ',' +
 C                              quot + '12/25/2002' + quot + ',' +
 C                                     '123456.78' +
 C                              ')')
```

SQL_MoveFirst

The **SQL_MoveFirst** API is used to move to the first record in the currently opened ADO recordset.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
0 = ADO/SQL database operation completed successfully

-1 = ADO/SQL database operation failed.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MoveFirst   Pr                  Like(TInt)
D Socket                             Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Move to first record
 *-------------------------------------------------------------------
C                   Eval      Rtn = SQL_MoveFirst(SQL_Socket)

C*                  ** Exit with Error Return - TCP Server Conne
C                   If        Rtn <> 0
C                   Exsr      $Exit
C                   Endif
```

SQL_MoveLast

The **SQL_MoveLast** API is used to move to the last record in the currently opened ADO recordset.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
0 = ADO/SQL database operation completed successfully

-1 = ADO/SQL database operation failed.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MoveLast    Pr                  Like(TInt)
D Socket                             Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Move to last record
 *-------------------------------------------------------------------
C                   Eval      Rtn = SQL_MoveLast(SQL_Socket)

C*                  ** Exit with Error Return - TCP Server Conne
C                   If        Rtn <> 0
C                   Exsr      $Exit
C                   Endif
```

SQL_MoveNext

The **SQL_MoveNext** API is used to move to the next record in the currently opened ADO recordset.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns

0 = ADO/SQL database operation completed successfully

-1 = ADO/SQL database operation failed.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MoveNext    Pr                  Like(TInt)
D Socket                             Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *---------------------------------------------------------------------
 *                  ** Move to next record
 *---------------------------------------------------------------------
C                   Eval      Rtn = SQL_MoveNext(SQL_Socket)

C*                  ** Exit with Error Return - TCP Server Conne
C                   If        Rtn <> 0
C                   Exsr      $Exit
C                   Endif
```

SQL_MovePrev

The **SQL_MovePrev** API is used to move to the previous record in the currently opened ADO recordset.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
0 = ADO/SQL database operation completed successfully

-1 = ADO/SQL database operation failed.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MovePrev    Pr                  Like(TInt)
D Socket                             Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Move to previous record
 *-------------------------------------------------------------------
C                   Eval      Rtn = SQL_MovePrev(SQL_Socket)

C*                  ** Exit with Error Return - TCP Server Conne
C                   If        Rtn <> 0
C                   Exsr      $Exit
C                   Endif
```

SQL_GetField

The **SQL_GetField** API is used to access the data values in any field from the current record in the currently opened ADO recordset.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns

The field data value is always returned in text format. The ILE programmer is required to convert any numeric values to the appropriate format in the ILE program by converting the text data to the appropriate format if required.

If an error occurs when retrieving a field, the text value ***ERROR*** is returned.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed.

RPG Prototype

```
DSQL_GetField     Pr                  Like(TString)
D Socket                              Like(TInt) Value
D FieldName                    256A   Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) FieldName - Pass the field name for a field from the current recordset.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get First Name field from current recordset
 *-------------------------------------------------------------------
C                   Eval      RtnField = SQL_GetField(SQL_Socket:
C                             'First')
```

SQL_GetFldChr

The **SQL_GetFieldChr** API is used to a text field value from the most recent **SQL_MoveFirst, SQL_MoveLast, SQL_MoveNext** or **SQL_MovePrev** call. This API has to go out to the RPG2SQL server to retrieve the field value.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
The field data value is returned in text format. If an error occurs when retrieving a field, the text value ***ERROR*** is returned.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldChr    Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
D FieldName                    256A   Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) FieldName - Pass the field name for a field from the current recordset.

Example Usage

```
 *------------------------------------------------------------------------
 *                  ** Get First Name field from current recordset
 *------------------------------------------------------------------------
C                    Eval      RtnField = SQL_GetFldChr(SQL_Socket:
C                              'First')
```

SQL_GetFldNum

The **SQL_GetFieldNum** API is used to a numeric field value from the most recent **SQL_MoveFirst, SQL_MoveLast, SQL_MoveNext** or **SQL_MovePrev** call. This API has to go out to the RPG2SQL server to retrieve the field value.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
The field data value is returned in numeric format. The return value is returned in **30.9 digit format**. Thirty positions with nine decimals. The RPG EVAL, MOVE or Z-ADD operations can be used to move the value to a smaller more focused RPG field size such as 7.2 or 15.2 or 9.0.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldNum    Pr            30P 9
D Socket                              Like(TInt) Value
D FieldName                    256A   Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) FieldName - Pass the field name for a field from the current recordset.

Example Usage

```
 *------------------------------------------------------------------
 *                  ** Get numeric field from current recordset
 *------------------------------------------------------------------
C*                  ** Numerics are always returned as packed 30,9.
C*                  ** If you need a smaller field such as 6,2, simply
C*                  ** evaluate the 6,2 field = to the returned number
C                   Eval      Dollars = SQL_GetFldNum(SQL_Socket:
C                             'Dollars')
```

SQL_GetFldDat

The **SQL_GetFieldDat** API is used to retrieve a date field value from the most recent **SQL_MoveFirst, SQL_MoveLast, SQL_MoveNext** or **SQL_MovePrev** call. This API has to go out to the RPG2SQL server to retrieve the field value.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
The field data value is returned in ISO date format as a timestamp value. RPG date functions can be used to reformat the ISO date as needed for display or other calculations.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldDat    Pr              Z
D Socket                              Like(TInt) Value
D FieldName                    256A   Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) FieldName - Pass the field name for a field from the current recordset.

Example Usage

```
 *------------------------------------------------------------------
 *                  ** Get date field from current recordset
 *------------------------------------------------------------------
C*                  ** Dates are always returned in ISO timestamp format
C                   Eval      Date1 = SQL_GetFldDat(SQL_Socket:
C                             'Date1')
```

SQL_SetSQLDelim

The **SQL_SetSQLDelim** API is used to set the delimiter that the RPG2SQL Integrator server software should use when returning delimited records.

Returns
0 = Delimiter was set successfully.

-1 = Error occurred while setting delimiter.

-2 = Unknown error.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_SetSQLDelim  Pr                  Like(Tint)
D Socket                              Like(TInt) Value
D Delim                         5A    Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Delim - Delimiter.

Example Usage

```
 *----------------------------------------------------------------
 *                  ** Set delimiter to 3 asterisks.
 *----------------------------------------------------------------
C                   Eval      Rtn = SQL_SetSQLDelim(SQL_Socket:
C                                                   '***')
```

SQL_MoveFirsBuf

The **SQL_MoveFirsBuf** API is used to move to the first record in the currently opened ADO recordset and return a delimited record buffer containing the record data. This command is used when high performance recordset reads are required.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
This function call returns a delimited record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MoveFirsBuf  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Go to first record in recordset and get
 *                  ** the record buffer data in return.
 *                  ** Requires only a single server pass for fast
 *                  ** performance.
 *-------------------------------------------------------------------
C                   Eval      RtnRecord=SQL_MoveFirsBuf(SQL_Socket)
```

SQL_MoveLastBuf

The **SQL_MoveLastBuf** API is used to move to the last record in the currently opened ADO recordset and return a delimited record buffer containing the record data. This command is used when high performance recordset reads are required.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
This function call returns a delimited record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MoveLastBuf  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Go to last record in recordset and get
 *                  ** the record buffer data in return.
 *                  ** Requires only a single server pass for fast
 *                  ** performance.
 *-------------------------------------------------------------------
C                   Eval      RtnRecord=SQL_MoveLastBuf(SQL_Socket)
```

SQL_MoveNextBuf

The **SQL_MoveNextBuf** API is used to move to the next record in the currently opened ADO recordset and return a delimited record buffer containing the record data. This command is used when high performance recordset reads are required.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
This function call returns a delimited record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MoveNextBuf  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *--------------------------------------------------------------------
 *                  ** Go to next record in recordset and get
 *                  ** the record buffer data in return.
 *                  ** Requires only a single server pass for fast
 *                  ** performance.
 *--------------------------------------------------------------------
C                   Eval      RtnRecord=SQL_MoveNextBuf(SQL_Socket)
```

SQL_MovePrevBuf

The **SQL_MovePrevBuf** API is used to move to the previous record in the currently opened ADO recordset and return a delimited record buffer containing the record data. This command is used when high performance recordset reads are required.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns

This function call returns a delimited record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_MovePrevBuf  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Go to previous record in recordset and get
 *                  ** the record buffer data in return.
 *                  ** Requires only a single server pass for fast
 *                  ** performance.
 *-------------------------------------------------------------------
C                   Eval      RtnRecord=SQL_MovePrevBuf(SQL_Socket)
```

SQL_GetFldChrB

The **SQL_GetFieldChrB** API is used to retrieve a text field value from the most recent **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** call.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
The field data value is returned in text format. If an error occurs when retrieving a field, the text value ***ERROR*** is returned.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldChrB   Pr                  Like(T4096String)
D RecordBuffer                4096A   Value
D                                     Varying
D Delimiter                   4096A   Value
D                                     Varying
D FldNumber                     10I 0 Value               D Delimiter
```

Parameters
1.) RecordBuffer - The delimited data record returned from the last **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** call.

2.) Delimiter - Record delimiter for returned record. Always Use tilde (~) when using **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** record operations.

3.) FldNumber - Enter the 1-based relative recordset field number.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get first field value in text format
 *-------------------------------------------------------------------
C                   Eval      RtnChar = SQL_GetFldChrB(
C                             RtnRecord:'~':1)
```

SQL_GetFldNumB

The **SQL_GetFldNumB** API is used to retrieve a numeric field value from the most recent **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** call.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
The field data value is returned in numeric format. The return value is returned in **30.9 digit format**. Thirty positions with nine decimals. The RPG EVAL, MOVE or Z-ADD operations can be used to move the value to a smaller more focused RPG field size such as 7.2 or 15.2 or 9.0.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldNumB   Pr            30P 9
D RecordBuffer                4096A   Value
D                                     Varying
D Delimiter                   4096A   Value
D                                     Varying
D FldNumber                     10I 0 Value
```

Parameters
1.) RecordBuffer - The delimited data record returned from the last **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** call.

2.) Delimiter - Record delimiter for returned record. Always Use tilde (~) when using **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** record operations.

3.) FldNumber - Enter the 1-based relative recordset field number.

Example Usage

```
 *------------------------------------------------------------------------
 *                  ** Get first field value in numeric format
 *------------------------------------------------------------------------
C                    Eval      RtnNum309 = SQL_GetFldNumB(
C                              RtnRecord:'~':1)
```

SQL_GetFldDatB

The **SQL_GetFldDatB** API is used to retrieve a date field value from the most recent **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** call.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
The field data value is returned in ISO date format as a timestamp value. RPG date functions can be used to reformat the ISO date as needed for display or other calculations.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldDatB    Pr              Z
D RecordBuffer                4096A   Value
D                                     Varying
D Delimiter                   4096A   Value
D                                     Varying
D FldNumber                     10I 0 Value                D Delimiter
```

Parameters
1.) RecordBuffer - The delimited data record returned from the last **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** call.

2.) Delimiter - Record delimiter for returned record. Always Use tilde (~) when using **SQL_MoveFirstBuf, SQL_MoveLastBuf, SQL_MoveNextBuf** or **SQL_MovePrevBuf** record operations.

3.) FldNumber - Enter the 1-based relative recordset field number.

Example Usage

```
 *----------------------------------------------------------------------
 *                  ** Get first field value in ISO date format
 *----------------------------------------------------------------------
C                   Eval      RtnDate = SQL_GetFldDatB(
C                             RtnRecord:'~':1)
```

SQL_GetStrRow

The **SQL_GetrStrRow** API is used to retrieve the current record buffer in delimited format. This API performs an implicit **SQL_MoveNext** after each call.

Use **SQL_GetFldChrB**, **SQL_GetNumB** or **SQL_GetDat2B** with a tilde (~) delimiter to parse the returned data record.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
This function call returns a delimited record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetStrRow     Pr                  Like(T4096String)
D Socket                               Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get Delimited Record data from SQL Table current record
 *                  ** and perform implicit SQL_MoveNext
 *-------------------------------------------------------------------
C                   Eval      RtnData = SQL_GetStrRow(SQL_Socket)
```

SQL_GetFldDat2B

The **SQL_GetFldDat2B** API is used to retrieve a date field value from the most recent **SQL_GetStrRow** call. This version should be used when SQL_GetStrRow is used to retrieve a record because date values are returned from the server MM/DD/YYYY format instead of ISO format.

Note: This API should be used instead of **SQL_GetFldDat** for retrieving date values after the SQL_GetStrRow API call

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns

The field data value is returned in ISO date format as a timestamp value. RPG date functions can be used to reformat the ISO date as needed for display or other calculations.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldDat2B  Pr              Z
D RecordBuffer                4096A   Value
D                                     Varying
D Delimiter                   4096A   Value
D                                     Varying
D FldNumber                     10I 0 Value               D Delimiter
```

Parameters

1.) RecordBuffer - The delimited data record returned from the last **SQL_GetStrRow** call.

2.) Delimiter - Record delimiter for returned record. Always Use tilde (~).

3.) FldNumber - Enter the 1-based relative recordset field number.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get first field value in ISO date format
 *-------------------------------------------------------------------
C                   Eval      RtnDate = SQL_GetFldDat2B(
C                             RtnRecord:'~':1)
```

SQL_GetDelimRcd

The **SQL_GetrDelimRcd** API is used to retrieve the current record buffer in delimited format after a **SQL_MoveFirst**, **SQL_MoveNext**, **SQL_MovePrev** or **SQL_MoveLast** database operation.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
This function call returns a delimited record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype
```
DSQL_GetDelimRcd  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
D Delimiter                           Like(TString) Value
D Quotes                              Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Delimiter - Record delimiter for returned record. Can be comma (**,**), tilde (~) or any other character you choose.

3.) Quotes - Use double quote values to surround text fields with. **0**=No quotes. **1**=Use double quotes.

Example Usage
```
 *----------------------------------------------------------------
 *                  ** Get Delimited Record data from SQL Table current record.
 *                  ** Use comma delimiter and do not surround text fields
 *                  ** with double quotes.
 *----------------------------------------------------------------
C                   Eval      RtnData = SQL_GetDelimRcd(SQL_Socket:
C                             ',':'0')
C*                  ** If errors, assume EOF reached or some
C*                  ** other error occurred. We're done reading recs.
C                   If        RtnRecord = '*ERROR*'
C                   Z-add     1             EOF1              1 0
C                   Endif
```

SQL_GetDelimFld

The **SQL_GetrDelimFld** API is used to retrieve a delimited field name list after recordset has been opened.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
This function call returns a delimited field name record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetDelimFld  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
D Delimiter                           Like(TString) Value
D Quotes                              Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Delimiter - Record delimiter for returned record. Can be comma (**,**), tilde (~) or any other character you choose.

3.) Quotes - Use double quote values to surround text fields with. **0**=No quotes. **1**=Use double quotes.

Example Usage

```
 *------------------------------------------------------------------------
 *                  ** Get Delimited Field List Record data from SQL Table
 *------------------------------------------------------------------------
C                    Eval      RtnData = SQL_GetDelimFld(SQL_Socket:
C                              ',':'0')
```

SQL_GetDelimFlt

The **SQL_GetrDelimFlt** API is used to retrieve a delimited field type list after recordset has been opened.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
This function call returns a delimited field name record buffer string or ***ERROR*** if an error occurs or if no record was found.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetDelimFlt  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
D Delimiter                           Like(TString) Value
D Quotes                              Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Delimiter - Record delimiter for returned record. Can be comma (**,**), tilde (~) or any other character you choose.

3.) Quotes - Use double quote values to surround text fields with. **0**=No quotes. **1**=Use double quotes.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get Delimited Field Type List data from SQL Table
 *-------------------------------------------------------------------
C                   Eval      RtnData = SQL_GetDelimFlt(SQL_Socket:
C                             ',':'0')
```

SQL_GetFldCount

The **SQL_GetFldCount** API is used to retrieve the number of fields in the current recordset.

Note: **SQL_RunSQLSel** must be used to open a recordset before this API can be used.

Returns
Count = This function call returns the number of fields found in the current recordset.

0 = No fields.

-2 = Unknown error.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_GetFldCount  Pr                  Like(TInt)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get number of fields in recordset
 *-------------------------------------------------------------------
C                   Eval      RtnCount = SQL_GetFldCount(SQL_Socket)
```

SQL_PCFileEmpty

The **SQL_PCFileEmpty** API creates an empty PC file on the server PC running the RPG2SQL Integrator server software.

Returns
0 = operation completed normally.

Not 0 = Errors occurred while deleting file.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_PCFileEmpty  Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D PCFile                              Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) PCFile - PC file name to create. Can be a standard file path name (C:\TEST.TXT) or UNC based file name (\\SERVER\SHARE\TEST.TXT).

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Create empty PC file
 *-------------------------------------------------------------------
C                   Eval      RtnErr = SQL_PCFileEmpty(SQL_Socket:'c:\test.txt')
```

SQL_PCFileExist

The **SQL_PCFileExist** API checks to see if a selected PC file exists on the server PC running the RPG2SQL Integrator server software.

Returns
0 = operation completed normally.

Not 0 = Errors occurred while deleting file.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_PCFileExist  Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D PCFile                              Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) PCFile - PC file name to check. Can be a standard file path name (C:\TEST.TXT) or UNC based file name (\\SERVER\SHARE\TEST.TXT).

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Check for PC file existence.
 *-------------------------------------------------------------------
C                   Eval      RtnErr = SQL_PCFileExist(SQL_Socket:'c:\test.txt')
```

SQL_PCFileRen

The **SQL_PCFileRen** API allows a selected PC file to be renamed on the server PC running the RPG2SQL Integrator server software.

Returns
0 = operation completed normally.

Not 0 = Errors occurred while deleting file.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_PCFileRen    Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D FromPCFile                          Like(TString) Value
D ToPCFile                            Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) FromPCFile - PC file name to rename from. Can be a standard file path name (C:\TEST.TXT) or UNC based file name (\\SERVER\SHARE\TEST.TXT).

3.) ToPCFile - PC file name to rename to. Can be a standard file path name (C:\TEST.TXT) or UNC based file name (\\SERVER\SHARE\TEST.TXT).

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Rename selected PC file
 *-------------------------------------------------------------------
C                   Eval      RtnErr = SQL_PCFileRen(SQL_Socket:
C                                                    'c:\test.txt':
C                                                    'c:\test2.txt')
```

SQL_PCFileCopy

The **SQL_PCFileCopy** API allows a selected PC file to be copied on the server PC running the RPG2SQL Integrator server software.

Returns

0 = operation completed normally.

Not 0 = Errors occurred while deleting file.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_PCFileCopy   Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D FromPCFile                          Like(TString) Value
D ToPCFile                            Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) FromPCFile - PC file name to copy. Can be a standard file path name (C:\TEST.TXT) or UNC based file name (\\SERVER\SHARE\TEST.TXT).

3.) ToPCFile - PC file name to copy to. Can be a standard file path name (C:\TEST.TXT) or UNC based file name (\\SERVER\SHARE\TEST.TXT).

Example Usage

```
 *------------------------------------------------------------------
 *                  ** Copy selected PC file
 *------------------------------------------------------------------
C                    Eval      RtnErr = SQL_PCFileCopy(SQL_Socket:
C                                                      'c:\test.txt':
C                                                      'e:\testdest.txt')
```

SQL_PCFileDel

The **SQL_PCFileDel** API allows a selected PC file to be deleted on the server PC running the RPG2SQL Integrator server software.

Returns
0 = operation completed normally.

Not 0 = Errors occurred while deleting file.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_PCFileDel    Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D PCFile                              Like(TString) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) PCFile - PC file name to delete. Can be a standard file path name (C:\TEST.TXT) or UNC based file name (\\SERVER\SHARE\TEST.TXT).

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Delete selected PC file
 *-------------------------------------------------------------------
C                   Eval      RtnErr = SQL_PCFileDel(SQL_Socket:'c:\test.txt')
```

SQL_LastErrNum

The **SQL_LastErrNum** API returns the last SQL error number that occurred on the RPG2SQL server in text format.

Note: SQL_LastErrNum, **SQL_LastErrNum2**, **SQL_LastErrMsg** and **SQL_LastFullErr** can be called as desired to retrieveinformation about the last error that occurred on the server. There is no rule as to which of these APIs must be used.

Returns
Text string value containing last error number. If an error occurs when retrieving the last error info, the text value ***ERROR*** is returned.

RPG Prototype
```
DSQL_LastErrNum    Pr                  Like(TString)
D Socket                               Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage
```
 *-------------------------------------------------------------------
 *                  ** Get last SQL error number
 *-------------------------------------------------------------------
C                   Eval      RtnErr = SQL_LastErrNum(SQL_Socket)
```

SQL_LastErrNum2

The **SQL_LastErrNum2** API returns the last SQL error number that occurred on the RPG2SQL server in packed numeric format.

Note: SQL_LastErrNum, **SQL_LastErrNum2**, **SQL_LastErrMsg** and **SQL_LastFullErr** can be called as desired to retrieveinformation about the last error that occurred on the server. There is no rule as to which of these APIs must be used.

Returns
A 15.0 numeric value containing the last error number.

RPG Prototype
```
DSQL_LastErrNum2   Pr            15P 0
D Socket                               Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get last SQL error number in packed format
 *-------------------------------------------------------------------
DRtnErr           S              15P 0

C                   Eval      RtnErr = SQL_LastErrNum2(SQL_Socket)
```

SQL_LastErrMsg

The **SQL_LastErrMsg** API returns the last SQL error message text from the last error that occurred on the RPG2SQL server. This text corresponds to the last error number which can be retrieved via the **SPL_LastErrNum** API.

Note: SQL_LastErrNum, **SQL_LastErrNum2**, **SQL_LastErrMsg** and **SQL_LastFullErr** can be called as desired to retrieveinformation about the last error that occurred on the server. There is no rule as to which of these APIs must be used.

Returns
Text string value containing last error number. If an error occurs when retrieving the last error info, the text value ***ERROR*** is returned.

RPG Prototype

```
DSQL_LastErrMsg   Pr                  Like(TString)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get last SQL error message text
 *-------------------------------------------------------------------
C                   Eval      RtnErr = SQL_LastErrMsg(SQL_Socket)
```

SQL_LastFullErr

The **SQL_LastFullErr** API returns a combination of the last SQL error message number and text from the RPG2SQL server.

Note: SQL_LastErrNum, **SQL_LastErrNum2**, **SQL_LastErrMsg** and **SQL_LastFullErr** can be called as desired to retrieveinformation about the last error that occurred on the server. There is no rule as to which of these APIs must be used.

Returns
Text string value containing last error number. If an error occurs when retrieving the last error info, the text value ***ERROR*** is returned.

RPG Prototype

```
DSQL_LastFullErr  Pr                  Like(TString)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Get last full SQL error message number and text
 *-------------------------------------------------------------------
C                    Eval      RtnErr = SQL_LastFullErr(SQL_Socket)
```

SQL_Quote

The **SQL_Quote** API accepts a string as an input value. If single quotes are found in the field value, they are padded with an additional single quote. This will allow the data value to get passed correctly to an SQL INSERT or UPDATE statement without needing to perform additional special editing on the string to check for single quotes in the data string. The function returns the newly quoted string value.

Returns
A string value where single quotes are padded with an additional single quote.

Example: **The People's Republic** will be returned as **The People''s Republic** after being passed into the SQL_Quote function.

RPG Prototype

```
DSQL_Quote        Pr          4096A   Varying
D iaInput                     4096A   Const
D                                     Varying
```

Parameters
1.) Variable length string value of up to 4096 characters.

Example Usage

```
 *-------------------------------------------------------------------
 *                  ** Check string for single quotes and pad if needed
 *-------------------------------------------------------------------
DWorkString       S            512A

C                    Eval      WorkString = SQL_Quote(WorkString)
```

SQL_RunSPBasic

The **SQL_RunSPBasic** API will run a SQL Server or other database stored procedure with input parameters allowed. The user must build the stored procedure call string before passing the single call string to call the stored procedure.

This function call is used for very simple stored procedure calls that do not return a recordset or parameters. No parameter values are returned, however a return code indicating success or failure is returned.

Returns
A return code indicating whether the stored procedure ran correctly.

0 = Stored procedure call completed normally.

Not 0 = Errors occurred while running stored procedure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_RunSPBasic   Pr                  Like(Tint)
D Socket                              Like(TInt) Value
D SPString                    4096A   Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.
2.) Variable length stored procedure execution string value of up to 4096 characters.

Example Usage

```
 *----------------------------------------------------------------------
 *                  ** Call the rjssample1 stored procedure
 *                  ** with no additional parameters.
 *----------------------------------------------------------------------
C                   Eval      Rtn = SQL_RunSPBasic(SQL_Socket:
C                             'rjssample1')
```

SQL_RunSP

The **SQL_RunSP** API will run a SQL Server or other database stored procedure with parameters as defined via the **SQL_AddParm** function. Multiple parameters can be added for each stored procedure call if needed.

This function call is used to call simple or complex stored procedures that can take input parameters as well as return output parameters if desired.

Note: No recordset will be returned from the **SQL_RunSP** call, however return parameters can be queried via the **SQL_GetParm** function. Use **SQL_RunSPSelect** if you will be returning parameters as well as a recordset from a stored procedure call.

SQL_ClearParms should be called to remove all stored procedure parameters before calling **SQL_AddParm** to define a new parameter.

Returns
A general return code indicating success or failure is returned, however the return parameters can also be queried for the stored procedure return code as well as any return parameters.

0 = Stored procedure call completed normally.

Not 0 = Errors occurred while running stored procedure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_RunSP        Pr                  Like(Tint)
D Socket                              Like(TInt) Value
D SPString                    4096A   Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.
2.) Variable length stored procedure execution string value of up to 4096 characters. Parameter values are set via SQL_AddParm function.

Sample SQL Server stored procedure: RJSSAMPLE1
This very simple stored procedure sets the stored procedure return code to 999 and sets the integer return value to 555 and the varchar value to 'This is a test'.

Note: The RJSSAMPLE1 stored procedure must be created on your SQL Server. You should create this sample stored procedure in the sample **pubs** database provided by Microsoft.

```
-------------------------------------------------
CREATE PROCEDURE rjssample1 @Return1  Int output,
                            @Return2  varchar(30) output as

 set @Return1 = 555
 set @Return2 = 'This is a test'
 return 999
-------------------------------------------------
```

Example Usage from RPG
The sample code listed below creates the return parameters for the stored procedure call, calls the RJSSAMPLE1 stored procedure and then retrieves the return parameter values.

```
C*                  ** Make sure stored procedure parameter list
C*                  ** is empty.
C                   Eval      Rtn = SQL_ClearParms(SQL_Socket)

C*                  ** This parm will hold the stored procedure
C*                  ** return code. Must always be defined first
C*                  ** if the SP return code will be used.
C*                  ** See RPGSQLH include for parm types and
C*                  ** ADO data types.
C                   Eval      Rtn = SQL_AddParm(SQL_Socket:
C                             'Return': adInteger :
C                             ParmReturnValue :
C                             0:' ')

C*                  ** This parm will hold the stored procedure
C*                  ** integer return field: Return1
C*                  ** See RPGSQLH include for parm types and
C*                  ** ADO data types.
C                   Eval      Rtn = SQL_AddParm(SQL_Socket:
C                             'Return1': adInteger :
C                             ParmOutput :
C                             0:' ')

C*                  ** This parm will hold the stored procedure
C*                  ** VarChar return field: Return2
C*                  ** See RPGSQLH include for parm types and
C*                  ** ADO data types.
C                   Eval      Rtn = SQL_AddParm(SQL_Socket:
C                             'Return2': adVarChar :
C                             ParmOutput :
C                             30:' ')

C*                  ** Call the stored procedure
C                   Eval      Rtn = SQL_RunSP(SQL_Socket:
C                             'rjssample1')

C*                  ** Retrieve SP Return Value to return record
C*                  ** SP Return Value is always converted to string.
C                   Eval      RtnParm = SQL_GetParm(SQL_Socket:
C                             'Return')

C*                  ** Retrieve Integer Return Value to return record
C*                  ** SP Return Value is always converted to string.
C                   Eval      RtnParm1 = SQL_GetParm(SQL_Socket:
C                             'Return1')

C*                  ** Retrieve VarChar Return Value to return record
C*                  ** SP Return Value is always converted to string.
C                   Eval      RtnParm2 = SQL_GetParm(SQL_Socket:
C                             'Return2')
```

SQL_RunSPSelect

The **SQL_RunSPSelect** API will run a SQL Server or other database stored procedure with parameters as defined via the **SQL_AddParm** function. Multiple parameters can be added for each stored procedure call if needed. This stored procedure call function also expects the stored procedure to return a recordset for processing.

This function should be used when the called stored procedure will be returning a recordset and potentially return parameters as well.

Note: A recordset will be returned from this call and return parameters can be queried via the **SQL_GetParm** function after the **SQL_DBCloseRS** function has been called. What this means is that you must read all records in the returned recordset before trying to read any return parameters. This is a Microsoft ADO/ODBC limitation.

SQL_ClearParms should be called to remove all store procedure parameters before calling SQL_AddParm to define a new parameter.

Returns
A general return code indicating success or failure is returned, however the return parameters can also be queried for the stored procedure return code as well as any return parameters.

0 = Stored procedure call completed normally.

Not 0 = Errors occurred while running stored procedure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_RunSPSelect  Pr                  Like(Tint)
D Socket                              Like(TInt) Value
D SPString                    4096A   Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.
2.) Variable length stored procedure execution string value of up to 4096 characters. Parameter values are set via SQL_AddParm function.

Sample SQL Server stored procedure: RJSSAMPLE2
This stored procedure runs a SQL record selection statement, sets the stored procedure return code to 999 and sets the integer value to 555 and the varchar value to 'This is a test'.

Note: The RJSSAMPLE2 stored procedure must be created on your SQL Server. You should create this sample stored procedure in the sample **pubs** database provided by Microsoft.

```
-------------------------------------------------
CREATE PROCEDURE rjssample2 @Return1  Int output,
                            @Return2  varchar(30) output as

select * from authors
set @Return1 = 555
set @Return2 = 'This is a test'
return 999
-------------------------------------------------
```

Example Usage from RPG
The sample code listed below creates the return parameters for the stored procedure call, calls the RJSSAMPLE2 stored procedure, reads only the first returned record, closes the recordset and then retrieves the return parameter values.

```
C

*                   ** Make sure stored procedure parameter list
C*                   ** is empty.
C                    Eval      Rtn = SQL_ClearParms(SQL_Socket)

C*                   ** This parm will hold the stored procedure
C*                   ** return code. Must always be defined first
C*                   ** if the SP return code will be used.
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return': adInteger :
C                              ParmReturnValue :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** integer return field: Return1
C*                   ** See RPGSQLH include for parm types and

C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return1': adInteger :
C                              ParmOutput :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** VarChar return field: Return2
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return2': adVarChar :
C                              ParmOutput :
C                              30:' ')

C*                   ** Call the stored procedure to return recordset
C                    Eval      Rtn = SQL_RunSPSelect(SQL_Socket:
C                              'rjssample2')

C*                   ** Retrieve First Record from Recordset
C*                   ** For this test we will only read one record
C*                   ** from the authors table in the pubs database.
C                    Eval      RtnRecord=SQL_MoveFirsBuf(SQL_Socket)

C*                   ** Close recordset if we're done reading from the
C*                   ** recordset.
C*                   ** The return parameters are available for reading
C*                   ** only after the recordset is closed. This is a
C*                   ** ADO command object limitation, not a RPG2SQL
C*                   ** limitation.
C                    Eval      Rtn=SQL_DBCloseRS(SQL_Socket)

C*                   ** Retrieve SP Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm = SQL_GetParm(SQL_Socket:
C                              'Return')

C*                   ** Retrieve Integer Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm1 = SQL_GetParm(SQL_Socket:
C                              'Return1')

C*                   ** Retrieve VarChar Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm2 = SQL_GetParm(SQL_Socket:
C                              'Return2')
```

SQL_ClearParms

The **SQL_ClearParms** API will reset the stored procedure parameter list. After clearing the parameter list, the SQL_AddParm function must be called to create a parameter list before a stored procedure can be called with parameters.

As good practice, **SQL_ClearParms** should be called prior to each stored procedure call to reset any input or return parameters that still exist from previous stored procedure calls. After calling **SQL_ClearParms**, SQL_AddParm should be used to set up any new parameters for the next stored procedure call.

Note: If **SQL_ClearParms** is not called before setting up a new parameters list, an incorrect number of parameters may exists for calling the next stored procedure, so always make sure to call SQL_ClearParms before adding new parameters and calling a stored procedure.

Returns
A general return code indicating success or failure is returned.

0 = Call completed normally.

Not 0 = Errors occurred while clearing parameters.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_ClearParms   Pr                  Like(TInt)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *                  ** Make sure stored procedure parameter list
 *                  ** is empty prior to adding parameters.
C                   Eval      Rtn = SQL_ClearParms(SQL_Socket)
```

SQL_AddParm

The **SQL_AddParm** API is used to add a parameter to the stored procedure parameter list prior to making a stored procedure call. This function can also set the input parameter value if the parameter will be used as input to a SQL stored procedure.

This function should be used when setting up a list of parameters to be sent to a stored procedure. If multiple parameters will be sent to a stored procedure, the SQL_AddParm should first be called to create the Return value parameter. Each additional call to SQL_AddParm should be made in the order which the selected parameter will be passed to or returned from the stored procedure call.

Note: As good practice, SQL_ClearParms should be called prior to each stored procedure call to reset any input or return parameters. If SQL_ClearParms is not called before setting up a new parameters list, an incorrect number of parameters may exists for calling the next stored procedure, so always make sure to call SQL_ClearParms before adding new parameters and calling a stored procedure.

Returns
A general return code indicating success or failure is returned.

0 = Parameter was added normally.

Not 0 = Errors occurred while adding stored procedure parameter.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_AddParm      Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D ParmName                     256    Value
D                                     Varying
D ParmDataType                  11P 0 Const
D ParmDirection                 11P 0 Const
D ParmLength                    11P 0 Const
D ParmValue                   4096    Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Parameter name - Enter a text name for the SQL Server parameter. The parameter name will be used when returning values from the stored procedure call if an output parameter is used. This value is required for both input and output parameters.

3.) Parameter data type - This setting specifies the data type that will be used when passing the parameter to the selected stored procedure. Use any of the following special constants to determine the data type.

adArray - Data is an array
adBigInt - Data is a large integer.
adBinary - Data contains binary data.
adBoolean - Data is a boolean true/false field.
adChar - Data is character.
adCurrency - Data is a currency field.
adDate - Data is a date field.
adDBDate - Data is a database data field.
adDBTime - Data is a database time field.
adDBTimeStamp - Data is a database timestamp field.
adDecimal - Data is a decimal field.
adDouble - Data is a double precision field.
adGUID - Data contains a GUID string.
adInteger - Data is in integer format.
adNumeric - Data is in numeric format.
adSingle - Data is in single precision format.
adSmallInt - Data is a small integer.
adVarChar - Data is a variable character string.
adVariant - Data is a variant field type.

4.) Parameter direction - This setting determines if the selected stored procedure parameter will be used for input, output or both. Use any of the following special constants to determine the parameter direction.

ParmInput - Input only parameter.
ParmOutput - Output only parameter.

ParmInputOutput - Input and Output Parameter
ParmReturnValue - Return code from stored procedure. If you will be expecting the stored procedure to return a return code, the first parameter passed to the stored procedure should be the return value.

5.) Parameter length - When sending to a text or character data type, set the parameter length to match the selected parameter field length. Set to 0 if the parameter will contain numeric data.

6.) Variable length parameter string value of up to 4096 characters. Set this value when sending an input value to the stored procedure for this parameter. Otherwise send the parameter out as blanks. If you will be sending numeric data to a stored procedure, it must be converted into a string buffer in your RPG program prior to sending the data to the **SQL_AddParm** function call.

Example Usage from RPG
The sample code listed below creates the return parameters for the stored procedure call, calls the RJSSAMPLE2 stored procedure, reads only the first returned record, closes the recordset and then retrieves the return parameter values.

```
*                   ** Make sure stored procedure parameter list
C*                   ** is empty.
C                    Eval      Rtn = SQL_ClearParms(SQL_Socket)

C*                   ** This parm will hold the stored procedure
C*                   ** return code. Must always be defined first
C*                   ** if the SP return code will be used.
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return': adInteger :
C                              ParmReturnValue :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** integer return field: Return1
C*                   ** See RPGSQLH include for parm types and

C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return1': adInteger :
C                              ParmOutput :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** VarChar return field: Return2
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return2': adVarChar :
C                              ParmOutput :
C                              30:' ')

C*                   ** Call the stored procedure to return recordset
C                    Eval      Rtn = SQL_RunSPSelect(SQL_Socket:
C                              'rjssample2')

C*                   ** Retrieve First Record from Recordset
C*                   ** For this test we will only read one record
C*                   ** from the authors table in the pubs database.
C                    Eval      RtnRecord=SQL_MoveFirsBuf(SQL_Socket)

C*                   ** Close recordset if we're done reading from the
C*                   ** recordset.
C*                   ** The return parameters are available for reading
C*                   ** only after the recordset is closed. This is a
C*                   ** ADO command object limitation, not a RPG2SQL
C*                   ** limitation.
C                    Eval      Rtn=SQL_DBCloseRS(SQL_Socket)

C*                   ** Retrieve SP Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm = SQL_GetParm(SQL_Socket:
C                              'Return')

C*                   ** Retrieve Integer Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm1 = SQL_GetParm(SQL_Socket:
C                              'Return1')

C*                   ** Retrieve VarChar Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm2 = SQL_GetParm(SQL_Socket:
C                              'Return2')
```

SQL_GetParm

The **SQL_GetParm** API is used to return a stored procedure return code value or to retrieve the value of a stored procedure output parameter. Output parameters allow stored procedures to return parameters other than the return code to a calling program.

Return codes and output parameters offer a nice way for SQL Server stored procedures to communicate with a calling program.

As good programming practice, make sure to first read the stored procedure return code and then retrieve all other return parameters as needed via the SQL_GetParm function.

Returns

A 4096 character text buffer that contains the returned stored procedure value or return code. If the returned value was a numeric value, you must convert the data into numeric format for use in your program or process the returned text value. ILE RPG has several built in functions for converting data from text to numeric.

RPG Prototype

```
DSQL_GetParm      Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
D ParmName                      256   Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Parameter name - Enter a text name for the SQL Server parameter. The parameter name will be used when returning values from the stored procedure call if an output parameter is used. This value is required for both input and output parameters.

Example Usage from RPG

The sample code listed below creates the return parameters for the stored procedure call, calls the RJSSAMPLE2 stored procedure, reads only the first returned record, closes the recordset and then retrieves the return parameter values.

```
*                   ** Make sure stored procedure parameter list
C*                   ** is empty.
C                    Eval      Rtn = SQL_ClearParms(SQL_Socket)

C*                   ** This parm will hold the stored procedure
C*                   ** return code. Must always be defined first
C*                   ** if the SP return code will be used.
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return': adInteger :
C                              ParmReturnValue :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** integer return field: Return1
C*                   ** See RPGSQLH include for parm types and

C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return1': adInteger :
C                              ParmOutput :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** VarChar return field: Return2
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return2': adVarChar :
C                              ParmOutput :
C                              30:' ')

C*                   ** Call the stored procedure to return recordset
C                    Eval      Rtn = SQL_RunSPSelect(SQL_Socket:
C                              'rjssample2')

C*                   ** Retrieve First Record from Recordset
C*                   ** For this test we will only read one record
C*                   ** from the authors table in the pubs database.
C                    Eval      RtnRecord=SQL_MoveFirsBuf(SQL_Socket)

C*                   ** Close recordset if we're done reading from the
C*                   ** recordset.
C*                   ** The return parameters are available for reading
C*                   ** only after the recordset is closed. This is a
C*                   ** ADO command object limitation, not a RPG2SQL
C*                   ** limitation.
C                    Eval      Rtn=SQL_DBCloseRS(SQL_Socket)

C*                   ** Retrieve SP Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm = SQL_GetParm(SQL_Socket:
C                              'Return')

C*                   ** Retrieve Integer Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm1 = SQL_GetParm(SQL_Socket:
C                              'Return1')

C*                   ** Retrieve VarChar Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm2 =
SQL_GetParm(SQL_Socket:

C                              'Return2')
```

SQL_DBCloseRS

The **SQL_DBCloseRS** API is used to close a recordset opened by running a SQL record select statement or stored procedure that creates or returns a recordset.

As good coding practice, the **SQL_DBCloseRS** function call should be called after you have completed working with a recordset, however the **SQL_DBCloseConn** function will also close all open tables as it closes the database connection if being used. Just make sure to call one of these functions before your program exits.

Note: When **SQL_RunSPSelect** is used to return a recordset from a stored procedure, all records must first be read and processed. After all records have been processed, **SQL_DBCloseRS** should be called and then the stored procedure return code and return parameters will be available via the **SQL_GetParm** function call.

Returns
A general return code indicating success or failure is returned.

0 = Parameter was added normally.

Not 0 = Errors occurred while adding stored procedure parameter.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DSQL_DBCloseRS    Pr                  Like(TInt)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

RPG2SQL ILE DDS Service Program Procedures

RPG2SQL ILE DDS Service Program Procedures

Introduction to DDS Handler Service Program

The RPG2SQL ILE DDS Service Program has been added to the RPG2SQL Integrator product to allow data returned from a SQL record operation to be quickly parsed into individual fields and placed into a DDS defined record structure or record format with a single API call.

Prior to the addition of this new service program, returning fields from an SQL query used to be a process of reading each individual field from the record buffer and manually moving the resulting data to a DDS or program defined field. Using the DDS Handler Service Program eases the task by returning multiple fields with a single ILE procedure call.

The SQLExtDDS and SQLExtRecDDS functions can be used from ILE RPG or COBOL programs to parse the record data returned by the SQL_MoveFirsBuf, SQL_MoveLastBuf, SQL_MoveNextBuf, SQL_MovePrevBuf procedures.

RPG Sample Program
SQTEST25 can be found in source file: **SOURCE** in library **RJSRPGSQL**.

COBOL Sample Program
SCTEST05 can be found in source file: **SOURCE** in library **RJSRPGSQL**.

SQLExtDDS

The **SQLExtDDS** API is used to read a DDS defined file layout into memory for use with the SQLExtRecDDS function. SQLExtRecDDS will parse the selected record buffer into multiple individual fields based on the DDS field definitions.

Returns
0 = Completed successfully

-9999 = Unknown error.

Nonzero - Some other error.

RPG Prototype

```
DSQLExtDDS        PR            10P 0
D DDSFile                     4096A   Value
D DDSLib                         *   Value
D FldDefs                        *   Value
D FldPtrs                        *   Value
D FldCount                       *
Value
```

Parameters
1.) DDSFile - The name of the AS/400 file layout to retrieve. The DDS layout can be used to retrieve a DDS layout for use as an externally defined data structure or to format data for writing to a file.

2.) DDSLib - The name of the AS/400 library for the file layout to retrieve.

3.) FldDefs - A pointer to an array that will hold the field definitions after SQLExtDDS has been called successfully. This parameter is passed to SQLExtRecDDS for use.

4.) FldPtrs - A pointer to a field pointer array that will hold the field definition pointers after SQLExtDDS has been called successfully. This parameter is passed to SQLExtRecDDS for use.

5.) FldCount - A pointer to a field where the DDS field count will be returned to. This parameter is passed to SQLExtRecDDS for use.

Example Usage from RPG

The sample code listed below retrieves the DDS layout for the QCUSTCDT file in library QIWS. The resulting return values are placed into the FldDefArray, FldPtrArray and FldDefCount return structures. The results of this call can later be passed to the SQL_ExtRecDDS API.

```
 *----------------------------------------------------------------
 *                  ** Get DDS Layout for selected AS/400 File
 *                  ** or externally defined data structure.
 *----------------------------------------------------------------
C                   Eval      INFILE = 'QCUSTCDT'
C                   Eval      INLIB  = 'QIWS'
C                   Eval      Rtn=SQL_ExtDDS(INFILE:INLIB:
C                                            %ADDR(FldDefArray):
C                                            %ADDR(FldPtrArray):
C                                            %ADDR(FldDefCount))
```

SQLExtRecDDS

The **SQLExtDDS** API is used to read a DDS defined file layout into memory for use with the SQLExtRecDDS function. SQLExtRecDDS will parse the selected record buffer into multiple individual fields based on the DDS field definitions.

Returns
0 = Completed successfully

-9999 = Unknown error.

Nonzero - Some other error.

RPG Prototype

```
DSQLExtDDS        PR            10P 0
D DDSFile                       10A   Const
D DDSLib                        10A   Const
D FldDefs                         *   Value
D FldPtrs                         *   Value
D FldCount                        *
Value
```

Parameters
1.) DDSFile - The name of the AS/400 file layout to retrieve. The DDS layout can be used to retrieve a DDS layout for use as an externally defined data structure or to format data for writing to a file.

2.) DDSLib - The name of the AS/400 library for the file layout to retrieve.

3.) FldDefs - A pointer to an array that will hold the field definitions after SQLExtDDS has been called successfully. This parameter is passed to SQLExtRecDDS for use.

4.) FldPtrs - A pointer to a field pointer array that will hold the field definition pointers after SQLExtDDS has been called successfully. This parameter is passed to SQLExtRecDDS for use.

5.) FldCount - A pointer to a field where the DDS field count will be returned to. This parameter is passed to SQLExtRecDDS for use.

Example Usage from RPG
The sample code listed below retrieves the DDS layout for the QCUSTCDT file in library QIWS. The resulting return values are placed into the FldDefArray, FldPtrArray and FldDefCount return structures. The results of this call can later be passed to the SQL_ExtRecDDS API.

```
 *---------------------------------------------------------------------
 *                  ** Get DDS Layout for selected AS/400 File
 *                  ** or externally defined data structure.
 *---------------------------------------------------------------------
C                   Eval      INFILE = 'QCUSTCDT'
C                   Eval      INLIB  = 'QIWS'
C                   Eval      Rtn=SQL_ExtDDS(INFILE:INLIB:
C                                            %ADDR(FldDefArray):
C                                            %ADDR(FldPtrArray):
C                                            %ADDR(FldDefCount))
```

The **SQLExtRecDDS** API is used to read a DDS defined file layout into memory for use with the SQLExtRecDDS function. SQLExtRecDDS will parse the selected record buffer into multiple individual fields based on the DDS field definitions.

Return codes and output parameters offer a nice way for SQL Server stored procedures to communicate with a calling program.

As good programming practice, make sure to first read the stored procedure return code and then retrieve all other return parameters as needed via the SQL_GetParm function.

Returns
A 4096 character text buffer that contains the returned stored procedure value or return code. If the returned value was a numeric value, you must convert the data into numeric format for use in your program or process the returned text value. ILE RPG has several built in functions for converting data from text to numeric.

RPG Prototype

```
DSQLExtDDS        PR            10P 0
D DDSFile                       10A   Const
D DDSLib                        10A   Const
D FldDefs                         *   Value
D FldPtrs                         *   Value
D FldCount                        *
Value

DSQLExtRecDDS     PR            10P 0
D RecordBuffer                4096A   Value
D RecordPtr                       *   Value
D**ecordSize                    10P 0 Value
D FldDefs                         *   Value
D FldPtrs                         *   Value
D FldCount                        *   Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Parameter name - Enter a text name for the SQL Server parameter. The parameter name will be used when returning values from the stored procedure call if an output parameter is used. This value is required for both input and output parameters.

Example Usage from RPG
The sample code listed below creates the return parameters for the stored procedure call, calls the RJSSAMPLE2 stored procedure, reads only the first returned record, closes the recordset and then retrieves the return parameter values.

```
*                   ** Make sure stored procedure parameter list
C*                   ** is empty.
C                    Eval      Rtn = SQL_ClearParms(SQL_Socket)

C*                   ** This parm will hold the stored procedure
C*                   ** return code. Must always be defined first
C*                   ** if the SP return code will be used.
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return': adInteger :
C                              ParmReturnValue :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** integer return field: Return1
C*                   ** See RPGSQLH include for parm types and

C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return1': adInteger :
C                              ParmOutput :
C                              0:' ')

C*                   ** This parm will hold the stored procedure
C*                   ** VarChar return field: Return2
C*                   ** See RPGSQLH include for parm types and
C*                   ** ADO data types.
C                    Eval      Rtn = SQL_AddParm(SQL_Socket:
C                              'Return2': adVarChar :
C                              ParmOutput :
C                              30:' ')

C*                   ** Call the stored procedure to return recordset
C                    Eval      Rtn = SQL_RunSPSelect(SQL_Socket:
C                              'rjssample2')

C*                   ** Retrieve First Record from Recordset
C*                   ** For this test we will only read one record
C*                   ** from the authors table in the pubs database.
C                    Eval      RtnRecord=SQL_MoveFirsBuf(SQL_Socket)

C*                   ** Close recordset if we're done reading from the
C*                   ** recordset.
C*                   ** The return parameters are available for reading
C*                   ** only after the recordset is closed. This is a
C*                   ** ADO command object limitation, not a RPG2SQL
C*                   ** limitation.
C                    Eval      Rtn=SQL_DBCloseRS(SQL_Socket)

C*                   ** Retrieve SP Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm = SQL_GetParm(SQL_Socket:
C                              'Return')

C*                   ** Retrieve Integer Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm1 = SQL_GetParm(SQL_Socket:
C                              'Return1')

C*                   ** Retrieve VarChar Return Value to return record
C*                   ** SP Return Value is always converted to string.
C                    Eval      RtnParm2 = SQL_GetParm(SQL_Socket:
C                              'Return2')
```

RPG2Excel ILE RPG Service Program Procedures

RPG2Excel ILE RPG Service Program Procedures

RPG2Excel Introduction

In response to several of our RPG2SQL Integrator customers using the RPG2SQL Integrator in conjunction with Microsoft Excel, RJS has added the ability to natively generate Excel workbooks from an RPG program.

Previously customers had to rely on using the outdated Microsoft Jet ODBC and Jet database drivers to write to Excel worksheets. While this works for creating simple data worksheets, using a database driver does not allow users to take full advantage of native Excel functionality.

The native RPG2Excel functionality allows RPG2SQL Integrator users to easily generate a brand new workbook, create a workbook from a template file, add data to a worksheet, select a range of cells and apply formatting, insert OLE objects into a worksheet or run specialized VBScript scripts when desired.

Excel functions such as those listed below:

Cell text formatting
Set cell data
Set fonts
Set underline
Set bold
Set italics
Setting calculation formulas

Inserting OLE Objects
Insert an org chart
Insert a Word document

VBScript Macro Capability
Running User Defined VBScript Macros

XLS_Launch

The **XLS_Launch** API is the second procedure to call after the SQL_Connect API is called to connect to the RPG2SQL Integrator PC component. This API launches Microsoft Excel on the PC running the RPG2SQL Integrator PC Server component. Once Excel has been launched on the PC Server, several Excel automation operations can be run including custom user defined VBScript code scripts if special functionality is needed.

If the **XLS_Launch** API completes successfully, a return code of 0 is returned from the PC server.

Returns
0 = Excel launch was successful.

-1 = Excel launch was not successful. Make sure Microsoft Excel is loaded on the the RPG2SQL Integrator server.

-2 = Unknown error.

-3 = Unknown communications error.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_Launch       Pr                  Like(Tint)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *---------------------------------------------------------------
 *                  ** Connect to RPG SQL Server and Launch Excel
 *---------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
C*                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif

C*                  ** Exit if Excel did not launch
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   If        Rtn <> 0
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Eval      *INLR = *On
C                   Return
C                   Endif
```

XLS_Visible

The **XLS_Visible** API is used to make Excel visible or to hide it. This API, if used at all, should be called after XLS_Launch.

Note: Excel visibility is off by default.

Returns
0 = Excel visibility was set successfully.

-1 = Excel visibility was not set successfully. Make sure Microsoft Excel is loaded on the the RPG2SQL Integrator server.

-2 =Unknown error.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_Visible      Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D Option                              Like(TInt) Value
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Option - Set Excel visibility. 0=hide Excel. 1=make Excel visible.

Example Usage

```
 *---------------------------------------------------------------
 *                  ** Make Excel visible.
 *---------------------------------------------------------------
C                   CallP     XLS_Visible(SQL_Socket:1)
```

XLS_Quit

The **XLS_Quit** API should be called when you are done working with Excel to generate spreadsheet data. This will shut down the current instance of Excel on the RPG2SQL Integrator server PC.

If the **XLS_Quit** API completes successfully, a return code of 0 is returned from the PC server.

Returns

0 = Excel shutdown was successful.

-1 = Excel shutdown was not successful. Make sure Microsoft Excel is loaded on the the RPG2SQL Integrator server.

-2 = Unknown error.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_Quit         Pr                  Like(TInt)
D Socket                              Like(TInt) Value
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Connect to RPG SQL Server and Launch Excel
 *                  ** and then quit right away.
 *-------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
C*                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif

C*                  ** Exit if Excel did not launch
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   If        Rtn <> 0
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Eval      *INLR = *On
C                   Return
C                   Endif

C*                  ** Quit Excel after launching it.
C                   Eval      Rtn = XLS_Quit(SQL_Socket)
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Endif

C*                  ** Bail out, we're done.
C                   Eval      *INLR = *On
C                   Return
```

XLS_Command

XLS_Command

The **XLS_Command** API is used to run several Excel functions. Functions not listed below can be run using VBScript code scripts. For an example of using VBScript in the XLS_Command API, refer tohttp://wiki.rjssoftware.com/wiki/index.php/Main_Page and then search for *XLS_RunScript*.

The XLS_Command API, if used at all, should be called after XLS_Launch.

Returns
0 = The Excel command ran successfully.

-1 = The Excel command did not run successfully. Make sure Microsoft Excel is loaded on the the RPG2SQL Integrator server.

-2 = Unknown error.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_Command      Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D XLCommand                    50A    Value
D                                     Varying
D ParmString                 4000A    Value
D                                     Varying
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.
2.) Excel command operation.
3.) Excel command parameters.

For specific example usage, see each command below.

CLEARRANGEALL

The **CLEARRANGEALL - XLS_Command** function is used to clear all the cells in the active worksheet.

Example to clear all cells.
```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'CLEARRANGEALL':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILECLOSE

The **FILECLOSE** - **XLS_Command** function is used to close the currently selected workbook.

Close the currently selected workbook.
```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'FILECLOSE':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILECLOSEALL

The **FILECLOSEALL** - **XLS_Command** function is used to close all open workbooks.

Close all workbooks.
```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'FILECLOSEALL':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILENEW

The **FILENEW** - **XLS_Command** function is used to generate a new empty workbook. The API can also generate a new workbook based on a template workbook as well.

Example to create a new empty workbook.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'FILENEW':'')
```

Example to create a new workbook based on the template name passed.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILENEW':'C:\EXCELTEMPLATE.XLS')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILEOPEN

The **FILEOPEN** - **XLS_Command** function is used to open an existing workbook.

Example to open an existing workbook.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILEOPEN':'C:\TEST.XLS')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILESAVE

The **FILESAVE** - **XLS_Command** function is used to save the currently open workbook. If the file has never been saved beforem use the **FILESAVEAS** function.

Example to save open workbook to same file name.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'FILESAVE':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILESAVEAS

The **FILESAVEAS** - **XLS_Command** function is used to save the currently open workbook to a new file name.

Example to save open workbook to c:\test2.xls using default Excel format.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILESAVEAS':'c:\test2.xls')
```

Example to save open workbook to c:\test2.dbf using Dbase IV format constant.

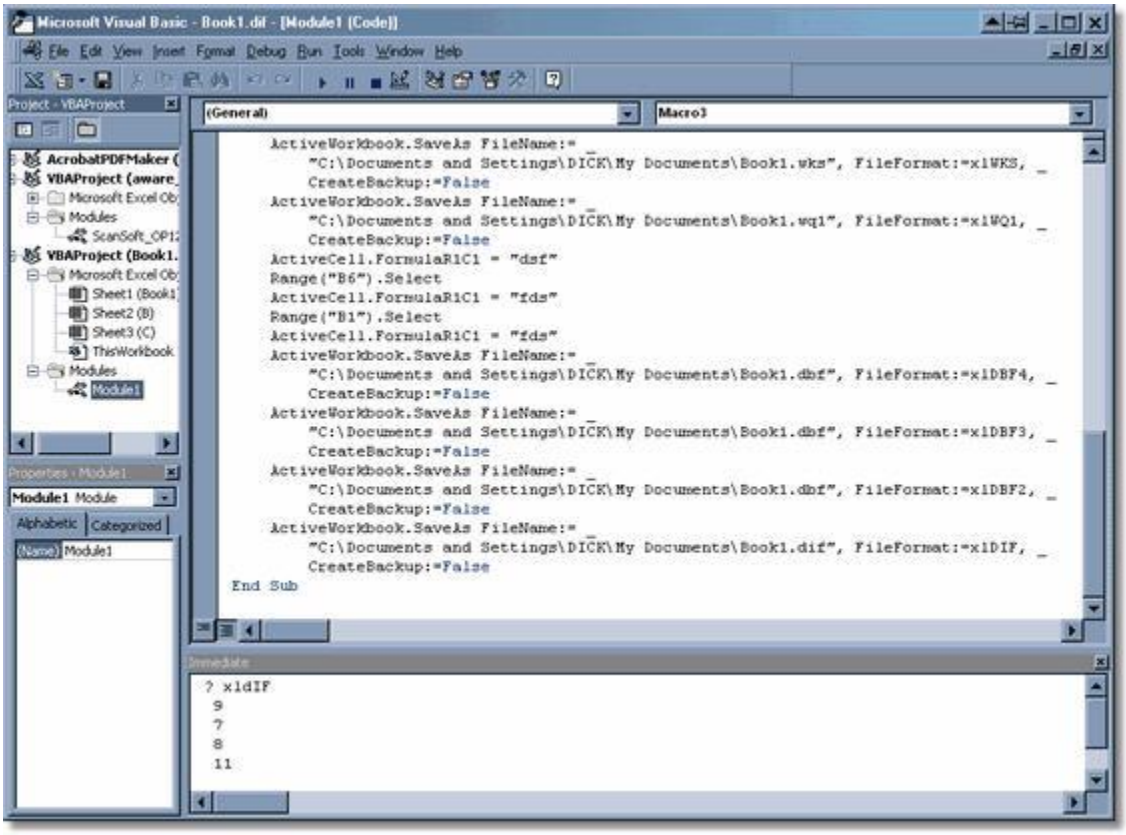
```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILESAVEAS':
C                             'C:\TEST.DBF~' +
C                             %trim(%EDITC(xlDBF4:'P')))
```

Example to save open workbook to c:\test2.dbf using Dbase IV format numeric value.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILESAVEAS':
C                             'C:\TEST.DBF~' +
C                             '11')
```

Listed below are some of the available Excel formats for saving a file. You can determine new output format value numbers as they are available in new versions of Excel by launching Excel, recording a new macro and then saving the workbook to the selected file type.

After recording the macro, review the macro code from Excel by pressing Alt-F11 to enter VB editing mode. Once you determine the new Excel content name you can paste it into the Excel immediate window to see its value.



Microsoft Excel VB Code Window (Alt-F11 from Excel).

Example: The value for xlDIF was displayed above by typing **? xlDIF** in the immediate window and pressing enter. The **xlDIF** value is **9**.

Available constants Microsoft Excel Output Format Constants

```
D** Microsoft Excel Constants
DxlNormal          C                   -4143
DxlSylk            C                   2
DxlText            C                   -4158
DxlTemplate        C                   17
DxlTextPrinter     C                   36
DxlExcel5          C                   39
DxlExcel9795       C                   43
DxlCSV             C                   6
DxlCSVMac          C                   22
DxlCSVMSDOS        C                   24
DxlCSVWindows      C                   23
DxlExcel4          C                   33
DxlExcel3          C                   29
DxlExcel2          C                   16
DxlWK3             C                   15
DxlWK3FM3          C                   32
DxlWK4             C                   38
DxlWK1FMT          C                   30
DxlWK1ALL          C                   31
DxlWK1             C                   5
DxlWKS             C                   4
DxlWQ1             C                   34
DxlDBF4            C                   11
DxlDBF3            C                   8
DxlDBF2            C                   7
DxlDIF             C                   9
DxlHTML            C                   44
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILEPRINT

The **FILEPRINT - XLS_Command** function is used to print selected worksheets.

Example to print selected worksheets.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'FILEPRINT':
C                             'HP LaserJet 2200 Series PCL 6')
```

Example to print two copies of selected worksheets.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'FILEPRINT':
C                             'HP LaserJet 2200 Series PCL 6~2')
```

Example to print two copies collated of selected worksheets.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'FILEPRINT':
C                             'HP LaserJet 2200 Series PCL 6~2~1')
```

P1 = Printer name
P2 = Number of copies
P3 = Collate (0=do not collate, 1=collate)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

FILEPRINTPREVIEW

The **FILEPRINTPREVIEW - XLS_Command** function is used to preview the printed view of selected worksheets.

Example to preview the printed view of selected worksheets.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILEPRINTPREVIEW')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETRANGEROWCOL

The **SETRANGEROWCOL - XLS_Command** function is used to select a cell range based on row/column in the worksheet.

Example to select cell A1

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETRANGEROWCOL':'1~1')
```

Example to select cells A1 to J1 (Row 1/Col1 to Row 1/Col 10)

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETRANGEROWCOL':'1~1~1~10')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELL

The **SETACTIVECELL - XLS_Command** function is used to set the active cell in the active worksheet using A1 format.

Example to set the active cell to A10. Once selected, the active cell operations can be used to set data or formatting for the cell.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETACTIVECELL':'A10')
```

P1 = Cell in A1 format

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLROWCOL

The **SETACTIVECELLROWCOL - XLS_Command** function is used to set the active cell in active worksheet via Row/Col

Example to set the active cell to A1. Row 1, Column 1. Once selected, the active cell operations can be used to set data or formatting for the cell.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETACTIVECELLROWCOL':'1~1')
```

P1 = Row Number
P2 = Column Number

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLVALUE

The **SETACTIVECELLVALUE - XLS_Command** function is used to set the active cell value.

Example to set the active cell to 'This is a test'.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETACTIVECELLVALUE':'This is a test')
```

P1 = Cellvalue

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFROMFILE

The **SETACTIVECELLFROMFILE - XLS_Command** function is used to set the active cell value from an ASCII text file.

Example to set value of active cell to the contents of a text file.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFROMFILE':'C:\TEST.TXT')
```

P1 = Text file name

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFORMULA

The **SETACTIVECELLFORMULA - XLS_Command** function is used to set the formula for the currently selected cell in A1 format.

Example to set cell formula to summarize cells A2 to A10

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFORMULA':'=SUM(A2:A10)')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFORMULAR1C1

The **SETACTIVECELLFORMULAR1C1** - **XLS_Command** function is used to set the formula for the selected cell in R1C1 format. See Excel documentation for details on the R1C1 format.

No example currently. Normally you should use the SETACTIVECELLFORMULA function.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTNAME

The **SETACTIVECELLFONTNAME - XLS_Command** function is used to set the active cell font name.

Example to set the active cell to 'Arial'.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTNAME':'Arial')
```

P1 = Font name

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTSIZE

The **SETACTIVECELLFONTSIZE - XLS_Command** function is used to set the active cell font size.

Example to set the active cell to 12pt.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSIZE':'12')
```

P1 = Font size

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTBOLD

The **SETACTIVECELLFONTBOLD** - **XLS_Command** function is used to set the bold value for the active cell.

Example to disable bold.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTBOLD':'0')
```

Example to set bold.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTBOLD':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTITALIC

The **SETACTIVECELLFONTITALIC** - **XLS_Command** function is used to set the italic value for the active cell.

Example to disable italics.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTITALIC':'0')
```

Example to set italics.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTITALIC':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFORECOLOR

The **SETACTIVECELLFORECOLOR** - **XLS_Command** function is used to set the foreground color for the active cell.

Example to set the font color to red.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFORECOLOR':'255')
```

Parameters
P1 = Numeric color value
Black=**0**
Blue=**16711680**
Cyan=**16776960**
Green=**65280**
Magenta=**16711935**
Red=**255**
White=**16777215**
Yellow=**65535**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLBACKCOLOR

The **SETACTIVECELLBACKCOLOR** - **XLS_Command** function is used to set the background color for the active cell.

Example to set the font color to red.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLBACKCOLOR':'255')
```

Parameters
P1 = Numeric color value
Black=**0**
Blue=**16711680**
Cyan=**16776960**
Green=**65280**
Magenta=**16711935**

Red=**255**
White=**16777215**
Yellow=**65535**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLMERGEANDCENTER

The **SETACTIVECELLMERGEANDCENTER** - **XLS_Command** function is used to set merge and center for the active cell. This will center any text within the active cell.

Example to merge and center the active cell.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLMERGEANDCENTER':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTSTRIKETHROUGH

The **SETACTIVECELLFONTSTRIKETHROUGH** - **XLS_Command** function is used to set the strikethrough value for the active cell.

Example to disable strikethrough.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSTRIKETHROUGH':'0')
```

Example to set strikethrough.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSTRIKETHROUGH':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTSUPERSCRIPT

The **SETACTIVECELLFONTSUPERSCRIPT** - **XLS_Command** function is used to set the superscript value for the active cell.

Example to disable superscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSUPERSCRIPT':'0')
```

Example to set superscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSUPERSCRIPT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTSUBSCRIPT

The **SETACTIVECELLFONTSUBSCRIPT** - **XLS_Command** function is used to set the subscript value for the active cell.

Example to disable subscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSUBSCRIPT':'0')
```

Example to set subscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSUBSCRIPT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTSHADOW

The **SETACTIVECELLFONTSHADOW** - **XLS_Command** function is used to set the shadow value for the active cell.

Example to disable shadow.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSHADOW':'0')
```

Example to set shadow.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTSHADOW':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTOUTLINEFONT

The **SETACTIVECELLFONTOUTLINEFONT** - **XLS_Command** function is used to set the outline font value for the active cell.

Example to disable outline fonts.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTOUTLINEFONT':'0')
```

Example to set outline fonts.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTOUTLINEFONT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLFONTUNDERLINE

The **SETACTIVECELLFONTUNDERLINE** - **XLS_Command** function is used to set the font underline value for the active cell.

Example to disable font underlining.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTUNDERLINE':'0')
```

Example to set font underlining to have a double underline.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLFONTUNDERLINE':'-4119')
```

Parameters
P1 = Numeric Font underline style
Single Underline = **2**
Double Underline = **-4119**
Single Accounting = **4**
Double Accounting = **5**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLSTYLE

The **SETACTIVECELLSTYLE** - **XLS_Command** function is used to set selection style formatting information for the active cell. A style contains a predefined list of formatting attributes. See the Excel documentation or the Format/Style menu for more detail on defining styles.

Example to set the style to Normal for the active cell.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLSTYLE':'Normal')
```

P1 = Style name. (See Excel docs for definition)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETACTIVECELLNUMBERFORMAT

The **SETACTIVECELLNUMBERFORMAT** - **XLS_Command** function is used to set numeric formatting information for active cell. See the Excel documentation or the Format/Cells/Number menu for more detail on defining numeric formats. Excel has a number of formatting options such as Currency, Accounting, General, etc.

Example to set the number format to Currency for all currently selected cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETACTIVECELLNUMBERFORMAT':
C                             '#,##0.00_);(#,##0.00)')
```

P1 = Selection number format (See Excel docs for definition)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETAUTOFITALL

The **SETAUTOFITALL - XLS_Command** function is used to select all cells that are defined in the range and automatically resize the columns to fit the data. You can define the cell range by using SETRANGESELECT or SETRANGEALL.

Example to select all cells and resize.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'SETAUTOFITALL':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETRANGEALL

The **SETRANGEALL - XLS_Command** function is used to select all cells in the active worksheet.

Example to select all cells.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:'SETRANGEALL':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETRANGEFORMULA

The **SETRANGEFORMULA** - **XLS_Command** function is used to set the formula for the selected cell in A1 format.

Example to set cell formula to summarize cells A2 to A10

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETRANGEFORMULA':'=SUM(A2:A10)')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETRANGESELECT

The **SETRANGESELECT** - **XLS_Command** function is used to select a cell range in the active worksheet.

Example to select cell A5

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETRANGESELECT':'A5')
```

Example to select cells A1 - B10

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETRANGESELECT':'A1:B10')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETRANGEVALUE

The **SETRANGEVALUE** - **XLS_Command** function is used to set the data value for an individual cell selection or range.

Example to set text data into the active cell or range.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETRANGEVALUE':'This is a test')
```

Example to set numeric data into the active cell or range.

```
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'SETRANGEVALUE':'200.00')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETCOLSELECTA1

The **SETCOLSELECTA1** - **XLS_Command** function is used to select column range using A1 format

Example to select columns A - D

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETCOLSELECTA1':'A~D')
```

P1 = Start Column Letter - (A,B,C,D,etc.)
P2 = End Column Letter - (A,B,C,D,etc.)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETROWSELECTA1

The **SETROWSELECTA1** - **XLS_Command** function is used to select row range using A1 format

Example to select rows 1 - 3

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETROWSELECTA1':'1~3')
```

P1 = Start Row Number - (1,2,3,etc.)
P2 = End Row Number - (1,2,3,etc.)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETCOLSELECTROWCOL

The **SETCOLSELECTROWCOL** - **XLS_Command** function is used to select column range using numeric column format.

Example to select columns A - D using relative numeric column position

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETCOLSELECTROWCOL':'1~4')
```

P1 = Start Column Number
P2 = End Column Number

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETROWSELECTROWCOL

The **SETROWSELECTROWCOL** - **XLS_Command** function is used to select row range using numeric format.

Example to select rows 5 - 10 using relative numeric column position

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETROWSELECTROWCOL':'5~10')
```

P1 = Start Row Number
P2 = End Row Number

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSCREENUPDATING

The **SETSCREENUPDATING** - **XLS_Command** function is used to set the Excel screen refresh option. Disabling screen refreshing can speed up the process when running worksheet operations.

Example to disable screen refreshing.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSCREENUPDATING':'0')
```

Example to enable screen refreshing.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSCREENUPDATING':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTSIZE

The **SETSELECTIONFONTSIZE** - **XLS_Command** function is used to set the Windows font size for the currently selected cell range.

Example to set font to 15pt.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSIZE':'15')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTNAME

The **SETSELECTIONFONTNAME** - **XLS_Command** function is used to set the Windows font name for the currently selected cell range.

Example to set font to Arial.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTNAME':'Arial')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTSUPERSCRIPT

The **SETSELECTIONFONTSUPERSCRIPT** - **XLS_Command** function is used to set the superscript value for the selected cell range.

Example to disable superscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSUPERSCRIPT':'0')
```

Example to set superscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSUPERSCRIPT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTSTRIKETHROUGH

The **SETSELECTIONFONTSTRIKETHROUGH** - **XLS_Command** function is used to set the strikethrough value for the selected cell range.

Example to disable strikethrough.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSTRIKETHROUGH':'0')
```

Example to set strikethrough.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSTRIKETHROUGH':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTSUBSCRIPT

The **SETSELECTIONFONTSUBSCRIPT** - **XLS_Command** function is used to set the subscript value for the selected cell range.

Example to disable subscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSUBSCRIPT':'0')
```

Example to set subscript.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSUBSCRIPT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTOUTLINEFONT

The **SETSELECTIONFONTOUTLINEFONT** - **XLS_Command** function is used to set the outline font value for the selected cell range.

Example to disable outline fonts.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTOUTLINEFONT':'0')
```

Example to set outline fonts.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTOUTLINEFONT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTSHADOW

The **SETSELECTIONFONTSHADOW** - **XLS_Command** function is used to set the font shadow value for the selected cell range.

Example to disable font shadowing.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSHADOW':'0')
```

Example to set font shadowing.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTSHADOW':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTBOLD

The **SETSELECTIONFONTBOLD** - **XLS_Command** function is used to set the font bold value for the selected cell range.

Example to disable font bolding.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTBOLD':'0')
```

Example to set font bolding.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTBOLD':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTITALIC

The **SETSELECTIONFONTITALIC** - **XLS_Command** function is used to set the font italic value for the selected cell range.

Example to disable font italics.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTITALIC':'0')
```

Example to set font italics.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTITALIC':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFONTUNDERLINE

The **SETSELECTIONFONTUNDERLINE** - **XLS_Command** function is used to set the font underline value for the selected cell range.

Example to disable font underlining.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTUNDERLINE':'0')
```

Example to set font underlining to have a double underline.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFONTUNDERLINE':'-4119')
```

Parameters
P1 = Numeric Font underline style
Single Underline = **2**
Double Underline = **-4119**
Single Accounting = **4**
Double Accounting = **5**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONBACKCOLOR

The **SETSELECTIONBACKCOLOR** - **XLS_Command** function is used to set the background color for the selected cell range.

Example to set the font color to red.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONBACKCOLOR':'255')
```

Parameters
P1 = Numeric color value
Black=**0**
Blue=**16711680**
Cyan=**16776960**
Green=**65280**
Magenta=**16711935**
Red=**255**
White=**16777215**

Yellow=**65535**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONFORECOLOR

The **SETSELECTIONFORECOLOR** - **XLS_Command** function is used to set the foreground color for the selected cell range.

Example to set the font color to red.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONFORECOLOR':'255')
```

Parameters
P1 = Numeric color value
Black=**0**
Blue=**16711680**
Cyan=**16776960**
Green=**65280**
Magenta=**16711935**
Red=**255**
White=**16777215**
Yellow=**65535**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONMERGEANDCENTER

The **SETSELECTIONMERGEANDCENTER** - **XLS_Command** function is used to set merge and center for cell range. This will merge all selected cells and center any text within the merged cells.

Example to merge and center selected cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONMERGEANDCENTER':'')
```

Note: Use this function last, after format and other changes, otherwise Excel may give unexpected results.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONSTYLE

The **SETSELECTIONSTYLE** - **XLS_Command** function is used to set selection style formatting information for cell range. A style contains a predefined list of formatting attributes. See the Excel documentation or the Format/Style menu for more detail on defining styles.

Example to set the style to Normal for all currently selected cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'SETSELECTIONSTYLE':
C                             'Normal')
```

P1 = Style name. (See Excel docs for definition)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONNUMBERFORMAT

The **SETSELECTIONNUMBERFORMAT** - **XLS_Command** function is used to set numeric formatting information for cell range. See the Excel documentation or the Format/Cells/Number menu for more detail on defining numeric formats. Excel has a number of formatting options such as Currency, Accounting, General, etc.

Example to set the number format to Currency for all currently selected cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONNUMBERFORMAT':'#,##0.00_);(#,##0.00)')
```

P1 = Selection number format (See Excel docs for definition)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONHORIZALIGN

The **SETSELECTIONHORIZALIGN** - **XLS_Command** function is used to set horizontal alignment for cell range selection.

Example to set the horizontal alignment to Center for all currently selected cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONHORIZALIGN':'-4108')
```

P1 = Numeric alignment value
General = **1**
Left = **-4131**
Right = **-4152**
Center = **-4108**
Justify = **-4130**
Fill = **5**
Center across selection = **7**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONVERTALIGN

The **SETSELECTIONVERTALIGN** - **XLS_Command** function is used to set vertical alignment for cell range selection.

Example to set the vertical alignment to Top for all currently selected cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONVERTALIGN':'-4160')
```

P1 = Numeric alignment value
Top = **-4160**
Center = **-4108**
Bottom = **-4107**
Justify = **-4130**

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONWRAPTEXT

The **SETSELECTIONWRAPTEXT** - **XLS_Command** function is used to set wrap text option for cell range selection.

Example to disable text wrapping.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONWRAPTEXT':'0')
```

Example to enable text wrapping

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONWRAPTEXT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONSHRINKTOFIT

The **SETSELECTIONSHRINKTOFIT** - **XLS_Command** function is used to set shrink to fit option for cell range selection.

Example to disable shrink to fit option.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONSHRINKTOFIT':'0')
```

Example to enable shrink to fit option.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONSHRINKTOFIT':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONMERGECELLS

The **SETSELECTIONMERGECELLS** - **XLS_Command** function is used to set merge cells option for cell range selection

Example to not merge cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONMERGECELLS':'0')
```

Example to merge cells.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONMERGECELLS':'1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTIONORIENTATION

The **SETSELECTIONORIENTATION** - **XLS_Command** function is used to set text orientation for cell range selection (90 to -90 degrees)

Example to set text rotation to 90 degrees.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTIONORIENTATION':'90')
```

P1 = Text orientation option in degrees (90 to -90)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTEDCOLDELETE

The **SETSELECTEDCOLDELETE** - **XLS_Command** function is used to delete currently selected columns.

Example to delete all currently selected coumns.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTEDCOLDELETE':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTEDROWDELETE

The **SETSELECTEDROWDELETE** - **XLS_Command** function is used to delete currently selected rows.

Example to delete all currently selected rows.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTEDROWDELETE':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETINSERTPAGEBREAK

The **SETINSERTPAGEBREAK** - **XLS_Command** function is used to insert page break in the worksheet at current cell location.

Example to insert page break.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'SETINSERTPAGEBREAK':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTEDBORDEROUTLINE

The **SETSELECTEDBORDEROUTLINE** - **XLS_Command** function is used to add an outline style border around all of the selected cells.

Example to add a solid, thick red line around the selected cells

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTEDBORDEROUTLINE':'1~4~255')
```

P1 = Line style
P2 = Line weight
P3 = Line color

Note: See **Line Styles for Border Operations** for descriptions of the style, weight and color values.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTEDBORDERLINE

The **SETSELECTEDBORDERLINE** - **XLS_Command** function is used to add a border to the selected cells. The border can surround the outer edge of all cells, the top edge, bottom edge, left edge, right edge or can be a diagonal line.

Example to add a solid, thick red line to the top edge of the selected cells

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTEDBORDERLINE':'8~1~4~255')
```

P1 = Border line type
P2 = Line style
P3 = Line weight
P4 = Line color

Note: See **Line Styles for Border Operations** for descriptions of the style, weight and color values.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

SETSELECTEDBORDERNONE

The **SETSELECTEDBORDERNONE** - **XLS_Command** function is used to remove any cell borders for cell range selection.

Example to remove all cell borders for selected cells

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'SETSELECTEDBORDERNONE':'')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

DISPLAYALERTS

The **DISPLAYALERTS** - **XLS_Command** function is used to set the display Excel alerts option. Alerts are messages that pop up if errors occur or if more information is required on a file open or save option, etc.

It is a good idea to disable alerts when creating workbooks on a server PC.

Example to disable alerts.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'DISPLAYALERTS':'0')
```

Example to enable alerts.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'DISPLAYALERTS':'1')
```

P1 = Alert option (0=no excel alerts,1=display excel alerts)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

WORKSHEETACTIVATE

The **WORKSHEETACTIVATE** - **XLS_Command** function is used to activate a worksheet by name.

Example to activate a worksheet named TEST1.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'WORKSHEETACTIVATE':'TEST1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

WORKSHEETADD

The **WORKSHEETADD** - **XLS_Command** function is used to add a new empty worksheet and optionally name the worksheet.

Example to insert a new worksheet without a special name.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'WORKSHEETADD':'')
```

Example to insert a new worksheet and name it TEST1.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'WORKSHEETADD':'TEST1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

WORKSHEETADDOLEOBJECT

The **WORKSHEETADDOLEOBJECT** - **XLS_Command** function is used to insert embedded OLE documents into the selected worksheet based on a saved file name.

Example to insert a Word document into the selected worksheet. No special coordinates are used for sizing, so they are all set to zero.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'WORKSHEETADDOLEOBJECT':'C:\TEST.DOC~0~0~0~0')')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

WORKSHEETDELETE

The **WORKSHEETDELETE** - **XLS_Command** function is used to delete a worksheet by name.

Example to delete a worksheet named TEST1.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'WORKSHEETDELETE':'TEST1')
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

WORKSHEETCOPY

The **WORKSHEETCOPY** - **XLS_Command** function is used to copy the active worksheet. You can specify its position relative to another worksheet or the beginning or end of the worksheets.

Example to copy active worksheet to the beginning.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'WORKSHEETCOPY':
C                             'BEFORE')
```

Example to copy active worksheet after worksheet Accounts and renamed Ledger.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'WORKSHEETCOPY':
C                             'AFTER~Accounts~Ledger')
```

P1 = Copy position (AFTER=either after P2 or at the end, BEFORE=either before P2 or at the beginning)
P2 = Worksheet name to position from
P3 = New worksheet name

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

EXCELDATEFORMATON

The **EXCELDATEFORMATON** - **XLS_Command** function is used to turn on Excel date formatting.

Example to turn on Excel date formatting. This will return the Excel date/time formatted as an ISO timestamp.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'EXCELDATEFORMATON':")
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

EXCELDATEFORMATOFF

The **EXCELDATEFORMATOFF** - **XLS_Command** function is used to turn off Excel date formatting.

Example to turn off Excel date formatting. This will return the Excel date/time as text.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'EXCELDATEFORMATOFF':")
```

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

Line Styles for Border Operations

Line Styles for Border Operations

Border operations allow users to outline a single sell or multiple cells. The following sections list the special constant values that can be used to define the look and feel for border operations.

Note: Special constants for the line styles, weights and colors are also defined in the RPGSQLH include source member

Border Line Type

This constant value helps define the type of outline that will be used within a cell range. An outline can be made around the entire cell or cell range, across the top, bottom, right of left sides.

EdgeTop=**8**
EdgeLeft=**7**
EdgeBottom=**9**
EdgeRight=**10**
DiagonalUp=**6**
DiagonalDown=**5**
InsideHorizontal=**12**
InsideVertical=**11**

Border Line Styles

This constant value helps define the style of line used when outlining a range of cells.

None = **-4142**
Continuous = **1**
Dash = **-4115**
DashDot = **4**
DashDotDot = **5**
Dot = **-4118**
Double = **-4119**
SlantDashDot = **13**

Border Weight

This constant value helps define the thickness or weight of the line used when outlining a range of cells.

Hairline=**1**
Medium=**-4138**
Thick=**4**

Thin=**2**

Border Color

This constant value helps define the color of line used when outlining a range of cells.

Black=**0**
Blue=**16711680**
Cyan=**16776960**
Green=**65280**
Magenta=**16711935**
Red=**255**
White=**16777215**
Yellow=**65535**

XLREFRESHPIVOTTABLE

The **XLREFRESHPIVOTTABLE** - **XLS_Command** function is used to refresh data in the selected pivot table.

Example to refresh a pivottable named: PIVOT1

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'XLREFRESHPIVOTTABLE':
C                             'PIVOT1')
```

Parameters
P1 = Pivot table name

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

XLSETCELLDATA

The **XLSETCELLDATA** - **XLS_Command** function is used to set values in selected cells.

Example to set the values in cells A1 and D3.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'XLSETCELLDATA':'A1~VALUE1~D3~VALUE2')
```

Parameters

P1 = Cell location 1
P2 = Value 1
through
P37 = Cell location 19
P38 = Value 19

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

XLSETCHECKBOX

The **XLSETCHECKBOX** - **XLS_Command** function is used to set values in selected checkboxes.

Example to select checkbox 1 and clear checkbox 3.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'XLSETCHECKBOX':'1~1~3~0')
```

Parameters
P1 = Checkbox 1
P2 = Value 1
through
P37 = Checkbox 19
P38 = Value 19

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

XLSSUBTOTAL

The **XLSSUBTOTAL** - **XLS_Command** function is used to apply auto subtotals.

Example to apply auto subtotals to the selection. There must be a column heading row in the selection. Here is an example of such a selection:

A	B	C	D
a	1	fred	10
a	2	vern	20
a	3	john	30
b	4	paul	40
b	5	ruth	50

c	6	bill	60

To group by the vaules in the first column and make totals for columns 2 and 4, the example is:

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'XLSSUBTOTAL':'1~2~4')
```

The result will be the like the following:

A	B	C	D
a	1	fred	10
a	2	vern	20
a	3	john	30
a Total	6		60
b	4	paul	40
b	5	ruth	50
b Total	9		90
c	6	bill	60
c Total	6		60
Grand Total	21		210

Parameters
P1 = Group by column
P2 - P11 = Columns to be subtotaled

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPRINTTITLEROWS

The **PAGESETUPPRINTTITLEROWS** - **XLS_Command** function is used to set the rows that contain the cells to be repeated at the top of each page, in A1 format.

Example to set the first 3 rows

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTTITLEROWS':'$1:$3')
```

P1 = Header rows in A1 format

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPRINTTITLECOLUMNS

The **PAGESETUPPRINTTITLECOLUMNS** - **XLS_Command** function is used to set the columns that contain the cells to be repeated on the left side of each page, in A1 format.

Example to set the first 3 columns

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTTITLECOLUMNS':'$A:$C')
```

P1 = Header columns in A1 format

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPRINTAREA

The **PAGESETUPPRINTAREA** - **XLS_Command** function is used to set the range to be printed, in A1 format.

Example to print cells A1 to C5

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'PAGESETUPPRINTAREA':
C                             '$A$1:$C$5')
```

P1 = Range to be printed in A1 format

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPLEFTHEADER

The **PAGESETUPLEFTHEADER** - **XLS_Command** function is used to set the left part of the header.

Example to set the left part of the header to the date.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPLEFTHEADER':'&D')
```

P1 =Text to use for the left part of the header. See Excel documentation for formatting codes that can be used in the header.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPCENTERHEADER

The **PAGESETUPCENTERHEADER** - **XLS_Command** function is used to set the center part of the header.

Example to set the center part of the header to the page number with hyphens around the number.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPCENTERHEADER':'- &P -')
```

P1 =Text to use for the center part of the header. See Excel documentation for formatting codes that can be used in the header.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPRIGHTHEADER

The **PAGESETUPRIGHTHEADER** - **XLS_Command** function is used to set the right part of the header.

Example to set the right part of the header to the file name.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPRIGHTHEADER':'&F')
```

P1 =Text to use for the right part of the header. See Excel documentation for formatting codes that can be used in the header.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPLEFTFOOTER

The **PAGESETUPLEFTFOOTER** - **XLS_Command** function is used to set the left part of the footer.

Example to set the left part of the footer to the date.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPLEFTFOOTER':'&D')
```

P1 =Text to use for the left part of the footer. See Excel documentation for formatting codes that can be used in the footer.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPCENTERFOOTER

The **PAGESETUPCENTERFOOTER** - **XLS_Command** function is used to set the center part of the footer.

Example to set the center part of the footer to the page number with hyphens around the number.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPCENTERFOOTER':'- &P -')
```

P1 =Text to use for the center part of the footer. See Excel documentation for formatting codes that can be used in the footer.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPRIGHTFOOTER

The **PAGESETUPRIGHTFOOTER** - **XLS_Command** function is used to set the right part of the footer.

Example to set the right part of the footer to the file name.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPRIGHTFOOTER':'&F')
```

P1 =Text to use for the right part of the footer. See Excel documentation for formatting codes that can be used in the footer.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPLEFTMARGIN

The **PAGESETUPLEFTMARGIN** - **XLS_Command** function is used to set the left margin, in inches.

Example to set the left the margin to 1.5 inches.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPLEFTMARGIN':'1.5')
```

P1 =Size of the left margin, in inches.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPRIGHTMARGIN

The **PAGESETUPRIGHTMARGIN** - **XLS_Command** function is used to set the right margin, in inches.

Example to set the right the margin to 1.5 inches.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPRIGHTMARGIN':'1.5')
```

P1 =Size of the right margin, in inches.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPTOPMARGIN

The **PAGESETUPTOPMARGIN** - **XLS_Command** function is used to set the top margin, in inches.

Example to set the top the margin to 1.5 inches.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPTOPMARGIN':'1.5')
```

P1 =Size of the top margin, in inches.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPBOTTOMMARGIN

The **PAGESETUPBOTTOMMARGIN** - **XLS_Command** function is used to set the bottom margin, in inches.

Example to set the bottom the margin to 1.5 inches.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPBOTTOMMARGIN':'1.5')
```

P1 =Size of the bottom margin, in inches.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPHEADERMARGIN

The **PAGESETUPHEADERMARGIN** - **XLS_Command** function is used to set the header margin, in inches.

Example to set the header the margin to .75 inches.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPHEADERMARGIN':'.75')
```

P1 =Size of the header margin, in inches.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPFOOTERMARGIN

The **PAGESETUPFOOTERMARGIN** - **XLS_Command** function is used to set the footer margin, in inches.

Example to set the footer the margin to .75 inches.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPFOOTERMARGIN':'.75')
```

P1 =Size of the footer margin, in inches.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPRINTHEADINGS

The **PAGESETUPPRINTHEADINGS** - **XLS_Command** function is used to set whether to print row and column headings.

Example to print row and column headings.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTHEADINGS':'1')
```

P1 =Whether to print row and column headings (1=print headings, 0=do not print headings)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPRINTGRIDLINES

The **PAGESETUPPRINTGRIDLINES** - **XLS_Command** function is used to set whether to print cell gridlines.

Example to print cell gridlines.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTGRIDLINES':'1')
```

P1 =Whether to print cell gridlines (1=print gridlines, 0=do not print gridlines)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPRINTCOMMENTS

The **PAGESETUPPRINTCOMMENTS** - **XLS_Command** function is used to set whether or how to print comments.

Example to print comments in place.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTCOMMENTS':'16')
```

Example to print comments as end notes.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTCOMMENTS':'1')
```

Example not to print comments.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTCOMMENTS':'-4142')
```

P1 =Whether or how to print comments

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPRINTQUALITY

The **PAGESETUPPRINTQUALITY** - **XLS_Command** function is used to set the print quality.

Example to set print quality to 1200 dpi on an HP printer.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPRINTQUALITY':'1200')
```

P1 =Print quality, the value of which may depend on the printer

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPCENTERHORIZONTALLY

The **PAGESETUPCENTERHORIZONTALLY** - **XLS_Command** function is used to set whether to center the data on the page horizontally.

Example to center the data on the page horizontally.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPCENTERHORIZONTALLY':'1')
```

P1 =Whether to center data horizontally (1=center, 0=do not center)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPCENTERVERTICALLY

The **PAGESETUPCENTERVERTICALLY** - **XLS_Command** function is used to set whether to center the data on the page vertically.

Example to center the data on the page vertically.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPCENTERVERTICALLY':'1')
```

P1 =Whether to center data vertically (1=center, 0=do not center)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPORIENTATION

The **PAGESETUPORIENTATION** - **XLS_Command** function is used to set the page orientation.

Example to set page orientation to portrait.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPORIENTATION':'1')
```

Example to set page orientation to landscape.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPORIENTATION':'2')
```

P1 =Page orientation

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPDRAFT

The **PAGESETUPDRAFT** - **XLS_Command** function is used to set whether to print graphics.

Example to print graphics.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'PAGESETUPDRAFT':'1')
```

P1 =Whether to print graphics (1=print graphics, 0=do not print graphics)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPPAPERSIZE

The **PAGESETUPPAPERSIZE** - **XLS_Command** function is used to set the paper size.

Example to set paper size to letter.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPAPERSIZE':'1')
```

Example to set paper size to legal.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPPAPERSIZE':'5')
```

P1 =Paper size. See Excel documentation for other values that you can use.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPFIRSTPAGENUMBER

The **PAGESETUPFIRSTPAGENUMBER** - **XLS_Command** function is used to set the first page number of a sheet.

Example to set the first page number of a sheet to 3.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPFIRSTPAGENUMBER':'3')
```

Example to have Excel determine the starting page of a sheet.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPFIRSTPAGENUMBER':'-4105')
```

P1 =First page number. See Excel documentation for further details.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPORDER

The **PAGESETUPORDER** - **XLS_Command** function is used to set the order of how pages are numbered and printed.

Example to set the order to down, then over.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'PAGESETUPORDER':'1')
```

Example to set the order to over, then down.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:'PAGESETUPORDER':'2')
```

P1 =Page order

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPBLACKANDWHITE

The **PAGESETUPBLACKANDWHITE** - **XLS_Command** function is used to set whether to print in black and white.

Example to print in black and white.

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPBLACKANDWHITE':'1')
```

P1 =Whether to print in black and white (1=print in black and white, 0=print in color, if supported)

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

PAGESETUPZOOM

The **PAGESETUPZOOM** - **XLS_Command** function is used to set the percentage (between 10 and 400 percent) by which Microsoft Excel will scale the selected worksheets for printing.

Example to scale the selected worksheets by 90 percent when printed

```
C                   Eval      Rtn= XLS_Command(SQL_Socket:
C                             'PAGESETUPZOOM':'90')
```

P1 =Zoom percentage

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

XLS_RowColCount

The **XLS_RowColCount** API is used to retrieve the number of rows and columns and the used range from the current Excel worksheet. This API is most useful to determine the limits of existing data when you are adding to a worksheet.

Note: Using this API requires at least V1.0.52 of the RPG2SQL Integrator server software.

Returns
This function call returns a delimited record buffer string or *ERROR* if an error occurs. The delimiter is a tilde (~). The values in the buffer are, first, the number of rows, then the number of columns, and, finally, the used range in A1 format.

Ex: If the worksheet has 4 rows and 10 columns, the value returned will be "4~10~A1:J4".

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg**, and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_RowColCount  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
```

Parameters
1.) Socket - SQL socket ID for RPG2SQL server connection.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Connect to RPG SQL Server and Launch Excel
 *-------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
 *                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
 *                  ** Exit if Excel did not launch
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   If        Rtn <> 0
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
 *                  ** Open existing workbook.
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                               'FILEOPEN':'C:\Test.xls')

 *-------------------------------------------------------------
 *                  ** Get row and column count from worksheet.
 *                  ** Extract the row and column counts and the
 *                  ** used range.
 *-------------------------------------------------------------
C                   Eval      Rtnrecord = XLS_RowColCount(SQL_Socket)
C                   Eval      rows = SQL_GetFldNum(RtnRecord:'~':1)
C                   Eval      columns = SQL_GetFldNum(RtnRecord:'~':2)
C                   Eval      usedRange = SQL_GetFldChr(RtnRecord:
C                                                       '~':3)

 *-------------------------------------------------------------------
 *                  ** Kill Excel
 *-------------------------------------------------------------------
                    Eval      Rtn = XLS_Quit(SQL_Socket)

 *-------------------------------------------------------------------
 *                  ** Disconnect from RPGSQL server
 *-------------------------------------------------------------------
C                   callp     SQL_Disconnect(SQL_Socket)
```

XLS_SetDelimRec

The **XLS_SetDelimRec** API is used to send a delimited iSeries data record buffer to the active Excel worksheet and then parse the data into multiple cells based on the delimiter passed to the API. Up to 256 values can be passed in the DelimRecord buffer, with a maximum total length of 4000.

The **XLS_SetDelimRec** API can be called multiple times as needed.

Returns
0 = Cells were set successfully.

-1 = Cells were not set successfully.

-2 = Unknown error.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_SetDelimRec  Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D StartRowNum                    7P 0 Value
D StartColNum                    7P 0 Value
D StartCell                      6A   Value
D                                     Varying
D DelimRecord                 4000A   Value
D                                     Varying
D DelimChar                      6A   Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) StartRowNum - This parameter is used to specify the row of the starting cell to which the delimited data record will be written.
Note: If this parameter is not used, the StartCell value must be specified and the StartRowNum and StartColNum values must be set to 0.

3.) StartColNum - This parameter is used to specify the column of the starting cell to which the delimited data record will be written.
Note: If this parameter is not used, the StartCell value must be specified and the StartRowNum and StartColNum values must be set to 0.

4.) StartCell - This parameter is used to specify the starting cell to which the delimited data record will be written.
Note: If this parameter is not used, the StartRowNum and StartColNum values must be greater than 0 and this value must be blanks.

5.) DelimRecord - This parameter is used to specify a delimited string value to be placed into the Excel worksheet and parsed into cells based on the selected delimiter value.

Ex: Test1, Test 2, Test3 would be placed into cells A1, B1 and C1 if **"A1"** was passed as the StartCell value.

6.) DelimChar - This character string is the delimiter used to parse the DelimRecord value into fields. If blanks are passed a comma is assumed.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Connect to RPG SQL Server and Launch Excel
 *-------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
C*                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
C*                  ** Exit if Excel did not launch
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   If        Rtn <> 0
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
C*                  ** Start new worksheet.
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILENEW':'')

 *-------------------------------------------------------------------
C*                  ** Send Test1, Test2, Test3 data strings
C*                  ** to cells A1, B1, C1
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_SetDelimRec(SQL_Socket:
C                             0:0:'A1':'Test1,Test2,Test3':',')

 *-------------------------------------------------------------------
C*                  ** Kill Excel
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Quit(SQL_Socket)

 *-------------------------------------------------------------------
C*                  ** Disconnect from RPGSQL server
 *-------------------------------------------------------------------
C                   callp     SQL_Disconnect(SQL_Socket)
```

XLS_SetDelimRc2

The **XLS_SetDelimRc2** API is used to send a delimited iSeries data record buffer to the active Excel worksheet and then parse the data into multiple cells based on the delimiter passed to the API. Up to 256 values can be passed in the DelimRecord buffer, with a maximum total length of 30,000.

The **XLS_SetDelimRc2** API can be called multiple times as needed.

Returns
0 = Cells were set successfully.

-1 = Cells were not set successfully.

-2 = Unknown error.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_SetDelimRc2 Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D StartRowNum                    7P 0 Value
D StartColNum                    7P 0 Value
D StartCell                      6A   Value
D                                     Varying
D DelimRecord                30000A   Value
D                                     Varying
D DelimChar                      6A   Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) StartRowNum - This parameter is used to specify the row of the starting cell to which the delimited data record will be written.
Note: If this parameter is not used, the StartCell value must be specified and the StartRowNum and StartColNum values must be set to 0.

3.) StartColNum - This parameter is used to specify the column of the starting cell to which the delimited data record will be written.
Note: If this parameter is not used, the StartCell value must be specified and the StartRowNum and StartColNum values must be set to 0.

4.) StartCell - This parameter is used to specify the starting cell to which the delimited data record will be written.
Note: If this parameter is not used, the StartRowNum and StartColNum values must be greater than 0 and this value must be blanks.

5.) DelimRecord - This parameter is used to specify a delimited string value to be placed into the Excel worksheet and parsed into cells based on the selected delimiter value.

Ex: Test1, Test 2, Test3 would be placed into cells A1, B1 and C1 if **"A1"** was passed as the StartCell value.

6.) DelimChar - This character string is the delimiter used to parse the DelimRecord value into fields. If blanks are passed a comma is assumed.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Connect to RPG SQL Server and Launch Excel
 *-------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
C*                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
C*                  ** Exit if Excel did not launch
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   If        Rtn <> 0
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
C*                  ** Start new workbook.
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILENEW':'')

 *-------------------------------------------------------------------
C*                  ** Send Test1, Test2, Test3 data strings
C*                  ** to cells A1, B1, C1
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_SetDelimRec(SQL_Socket:
C                             0:0:'A1':'Test1,Test2,Test3':',')

 *-------------------------------------------------------------------
C*                  ** Kill Excel
 *-------------------------------------------------------------------
                    Eval      Rtn = XLS_Quit(SQL_Socket)

 *-------------------------------------------------------------------
C*                  ** Disconnect from RPGSQL server
 *-------------------------------------------------------------------
C                   callp     SQL_Disconnect(SQL_Socket)
```

XLS_GetDelimRec

The **XLS_GetDelimRec** API is used to retrieve a delimited iSeries data record buffer from the active Excel worksheet. The amount of data that can be received is up to 4096 characters total length.

The **XLS_GetDelimRec** API can be called multiple times, as needed.

Returns

This function call returns a delimited record buffer string or *ERROR* if an error occurs.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_GetDelimRec  Pr                  Like(T4096String)
D Socket                              Like(TInt) Value
D StartRowNum                   7P 0  Value
D StartColNum                   7P 0  Value
D EndColNum                     7P 0  Value
D CellRange                    20A    Value
D                                     Varying
D DelimParms                 4000A    Value
D                                     Varying
D DelimChar                     6A    Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) StartRowNum - This parameter is used to specify the starting worksheet row from which the data will be read.
Note: If this parameter is not used, the CellRange value must be specified and the StartRowNum, StartColNum, and EndColNum values must be set to 0.

3.) StartColNum - This parameter is used to specify the starting worksheet column of the range from which the data will be read.
Note: If this parameter is not used, the CellRange value must be specified and the StartRowNum, StartColNum, and EndColNum values must be set to 0.

4.) EndColNum - This parameter is used to specify the ending worksheet column of the range from which the data will be read.
Note: If this parameter is not used, the CellRange value must be specified and the StartRowNum, StartColNum, and EndColNum values must be set to 0.

5.) CellRange - This parameter is used to specify the single-row worksheet range from which the data will be read.
Note: If this parameter is not used, the StartRowNum, StartColNum, and EndColNum values must be greater than 0 and this value must be blanks.

6.) DelimParms - This parameter is not used. Leave it blank.

7.) DelimChar - This parameter is the character string used to delimit the values in the record buffer. If blanks are passed a tilde (~) is assumed.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Connect to RPG SQL Server and Launch Excel
 *-------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
 *                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
 *                  ** Exit if Excel did not launch
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   If        Rtn <> 0
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
 *                  ** Open existing workbook.
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                               'FILEOPEN':'C:\Test.xls')

 *-------------------------------------------------------------
 *                  ** Read second row from worksheet, columns
 *                  ** 1 to 13. Use a semicolon as the delimiter.
 *                  ** Use row and column values.
 *-------------------------------------------------------------
C                   Eval      Rtnrecord = XLS_GetDelimRec(SQL_Socket:
C                                         2:1:13:'':'':';')

 *-------------------------------------------------------------
 *                  ** Read third row from worksheet, columns
 *                  ** 1 to 13. Use 2 asterisks as the delimiter.
 *                  ** Use A1 format.
 *-------------------------------------------------------------
C                   Eval      Rtnrecord = XLS_GetDelimRec(SQL_Socket:
C                                         0:0:0:'A3:M3':'':'**')

 *-------------------------------------------------------------------
 *                  ** Kill Excel
 *-------------------------------------------------------------------
                    Eval      Rtn = XLS_Quit(SQL_Socket)

 *-------------------------------------------------------------------
 *                  ** Disconnect from RPGSQL server
 *-------------------------------------------------------------------
C                   callp     SQL_Disconnect(SQL_Socket)
```

XLS_GetDelimRc2

The **XLS_GetDelimRc2** API is used to retrieve a delimited iSeries data record buffer from the active Excel worksheet. The amount of data that can be received is up to 30,000 characters total length.

The **XLS_GetDelimRc2** API can be called multiple times, as needed.

Returns
This function call returns a delimited record buffer string or *ERROR* if an error occurs.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg**, and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_GetDelimRc2  Pr                  Like(T30000String)
D Socket                              Like(TInt) Value
D StartRowNum                   7P 0  Value
D StartColNum                   7P 0  Value
D EndColNum                     7P 0  Value
D CellRange                    20A    Value
D                                     Varying
D DelimParms                30000A    Value
D                                     Varying
D DelimChar                     6A    Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) StartRowNum - This parameter is used to specify the starting worksheet row from which the data will be read.
Note: If this parameter is not used, the CellRange value must be specified and the StartRowNum, StartColNum, and EndColNum values must be set to 0.

3.) StartColNum - This parameter is used to specify the starting worksheet column of the range from which the data will be read.
Note: If this parameter is not used, the CellRange value must be specified and the StartRowNum, StartColNum, and EndColNum values must be set to 0.

4.) EndColNum - This parameter is used to specify the ending worksheet column of the range from which the data will be read.
Note: If this parameter is not used, the CellRange value must be specified and the StartRowNum, StartColNum, and EndColNum values must be set to 0.

5.) CellRange - This parameter is used to specify the single-row worksheet range from which the data will be read.
Note: If this parameter is not used, the StartRowNum, StartColNum, and EndColNum values must be greater than 0 and this value must be blanks.

6.) DelimParms - This parameter is not used. Leave it blank.

7.) DelimChar - This parameter is the character string used to delimit the values in the record buffer. If blanks are passed a tilde (~) is assumed.

Example Usage

```
 *-------------------------------------------------------------
 *                  ** Connect to RPG SQL Server and Launch Excel
 *-------------------------------------------------------------
C                   Eval      SQL_Socket = SQL_Connect('10.1.1.1')
 *                  ** Exit with Error Return - TCP Server Connect
C                   If        SQL_Socket = -999
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
 *                  ** Exit if Excel did not launch
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   If        Rtn <> 0
C                   callp     SQL_Disconnect(SQL_Socket)
C                   Eval      *INLR = *On
C                   Return
C                   Endif

 *-------------------------------------------------------------------
 *                  ** Open existing workbook.
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                               'FILEOPEN':'C:\Test.xls')

 *-------------------------------------------------------------
 *                  ** Read second row from worksheet, columns
 *                  ** 1 to 13. Use a semicolon as the delimiter.
 *                  ** Use row and column values.
 *-------------------------------------------------------------
C                   Eval      Rtnrecord = XLS_GetDelimRc2(SQL_Socket:
C                                         2:1:13:'':'':';')

 *-------------------------------------------------------------
 *                  ** Read third row from worksheet, columns
 *                  ** 1 to 13. Use 2 asterisks as the delimiter.
 *                  ** Use A1 format.
 *-------------------------------------------------------------
C                   Eval      Rtnrecord = XLS_GetDelimRc2(SQL_Socket:
C                                         0:0:0:'A3:M3':'':'**')

 *-------------------------------------------------------------------
 *                  ** Kill Excel
 *-------------------------------------------------------------------
                    Eval      Rtn = XLS_Quit(SQL_Socket)

 *-------------------------------------------------------------------
 *                  ** Disconnect from RPGSQL server
 *-------------------------------------------------------------------
C                   callp     SQL_Disconnect(SQL_Socket)
```

XLS_RunScript

The **XLS_RunScript** API is useful for users who have customized scripting needs when creating workbooks. Users can write scripts using VBScript that can talk to the Microsoft Excel OLE object model to automate selected Excel workbook creation processes not covered by the general RPG2Excel API functionality.

If the **XLS_RunScript** API completes successfully, a return code of 0 is returned from the PC server.

Returns
0 = Excel script run was successful.

-1 = Excel script run was not successful.

-2 = Unknown error.

-3 = Unknown communications failure.

Note: You can use the **SQL_LastErrNum**, **SQL_LastErrMsg** and **SQL_LastFullErr** APIs to determine why the API call failed if an error is returned.

RPG Prototype

```
DXLS_RunScript    Pr                  Like(TInt)
D Socket                              Like(TInt) Value
D ScriptFile                   255    Value
D                                     Varying
D FunctionName                 100    Value
D                                     Varying
D ParmString                  3700    Value
D                                     Varying
```

Parameters

1.) Socket - SQL socket ID for RPG2SQL server connection.

2.) Script file - Location of the PC script file to run.

3.) Function name - The name of the VBScript function to call within the selected script file. Functions can have all kind of names depending on your scripting requirements.

4.) Parm string - This parameter can be used to pass delimited data strings to the selected VBScript if parameters need to be passed to the script for processing within the script.

Example Usage

1.) Type the following VBScript code in a text editor on the PC where RPG2SQL Integrator is running and save the code as C:\temp\helloexcel.vbs.

```
Function TestExcel1(sCommand,sExcelObject,sObject2,sObject3)
    '---------------------------------------------------------------------
    'Function: TestExcel1
    'Purpose : Write a few cells of data into the current Excel sheet.
    'Parms   : Single parameter separated by semicolons
    '---------------------------------------------------------------------

    '---------------------------------
    'Break down parameters. Parms should be delimited by semicolons if passed.
    'Parms are all zero-baed (Parm1 = arrparms(0), Parm2=arrparms(1), etc..)
    '---------------------------------
    arrParms=Split(sCommand,";")

    '---------------------------------
    'Move parm 1 value to cell B3
    '---------------------------------
    sExcelObject.ActiveSheet.Range("B3").value = arrparms(0)

    '---------------------------------
    'Move parm 2 value to cell C3
    '---------------------------------
    sExcelObject.ActiveSheet.Range("C3").value = arrparms(1)

    TestExcel1 = 0
End Function
```

2.) The following code runs the script you created in step 1.

```
 *-------------------------------------------------------------------
 *                  ** Launch MS Excel and make it visible
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Launch(SQL_Socket)
C                   Eval      Rtn = XLS_Visible(SQL_Socket:1)
 *-------------------------------------------------------------------
 *                  ** Create new spreadsheet file
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_Command(SQL_Socket:
C                             'FILENEW':'')
 *-------------------------------------------------------------------
 *                  ** Run Excel VBScript  Macro to set data in MS
 *                  ** Excel spreadsheet cells.
 *-------------------------------------------------------------------
C                   Eval      Rtn = XLS_RunScript(SQL_Socket:
C                             'c:\temp\helloexcel.vbs':
C                             'TestExcel1':
C                             'Call Test Data for B3;' +
C                             'Call Test Data for C3')
```

Problem Handling

Problem Handling

Common Errors

If errors occur while running the iSeries RPG2SQL Integrator APIs, always make sure to check the AS/400 job log by using the DSPJOBLOG command from the AS/400 command line. Many times the job log will give some idea as to what is failing.

You can also run the RPG2SQL Integrator PC Server component in debug mode if you want to see what errors are occurring.

If you need to see what's happening you can also turn on logging in the settings for the RPG2SQL Integrator PC component.

Other Errors

If you cannot resolve the problem you're experiencing by using the previous troubleshooting methods, you should contact RJS Software Systems for advice on solving the problems you're experiencing.

Appendix A: RPG2SQL ILE COBOL Service Program Procedures

Appendix A: RPG2SQL ILE COBOL Service Program Procedures

COBOL Working Storage Variables

The following variables should be included in all COBOL programs you create that use the RPG2SQL Integrator. You can add these variables to a COBOL copy book is desired to avoid redundancy.

COBOL example usage:

```
*----------------------------------
*** RPG2SQL Work variables
*----------------------------------
 01 SQL_SOCKET   PIC  9999999999 COMP-3.
 01 SQL_IPADDR   PIC  X(100).
 01 SQL_DATASRC  PIC  X(256).
 01 SQL_SQLSEL   PIC  X(4096).
 01 SQL_SQLEXEC  PIC  X(4096).
 01 SQL_RECORD   PIC  X(4096).
 01 SQL_RTN      PIC  9999999999 COMP-3.
 01 SQL_INSTR    PIC  X(4096).
 01 SQL_ADDSTR   PIC  X(4096).
 01 SQL_OUTSTR   PIC  X(4096).
 01 SQL_FLDNAME  PIC  X(256).

*----------------------------------
*** RPG2SQL Return Work variables
*----------------------------------
 01 WORKCHAR     PIC  X(256).
 01 WORKNUM      PIC  S9(21)V9(9) COMP-3.
 01 WORKDATE     FORMAT TIMESTAMP.
```

SQLConnect

This function is the COBOL version of the SQL_Connect procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Connect to RPG2SQL PC Server
*    ** Returne IP socket ID for RPG2SQL
*    ** PC server
*    ---------------------------------
     CALL PROCEDURE 'SQLCONNECT'
          USING SQL_IPADDR RETURNING SQL_SOCKET.

*    ** Exit program if no IP connection made
           IF SQL_SOCKET <= -999
              EXIT PROGRAM AND CONTINUE RUN UNIT
           END-IF.
```

SQLDisconnect

This function is the COBOL version of the SQL_Disconnect procedure.

COBOL example usage:

```
*    ----------------------------------
*    ** Disconnect from RPG2SQL PC Server
*    ----------------------------------
     CALL PROCEDURE 'SQLDISCONNECT'
          USING SQL_SOCKET RETURNING SQL_RTN.
```

SQLTimeout

This function is the COBOL version of the SQL_Timeout procedure.

COBOL example usage:

```
*    ----------------------------------
*    ** Set SQL Timeout to 60 seconds
*    ----------------------------------
     MOVE 60 to SQL_TIMEOUTVAL.

     CALL PROCEDURE 'SQLTIMEOUT'
          USING SQL_SOCKET SQL_TIMEOUTVAL RETURNING SQL_RTN.
```

SQLDBOpenConn

This function is the COBOL version of the SQL_DBOpenConn procedure.

COBOL example usage:

```
*    ----------------------------------
*    ** Set ADO Data Source for Access Database
*    ----------------------------------
     MOVE 'Driver={Microsoft Access Driver (*.mdb)}; DBQ=C:\Progra
-    'm files\rpgsqlsv\irpgsql.mdb; Uid=Admin Pwd=;'
     to SQL_DATASRC.

*    ----------------------------------
*    ** Open Database Connection
*    ** to selected ADO/ODBC database
*    ----------------------------------
     CALL PROCEDURE 'SQLDBOPENCONN'
               USING SQL_SOCKET SQL_DATASRC RETURNING SQL_RTN.
```

SQLDBCloseConn

This function is the COBOL version of the SQL_DBCloseConn procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Close Database Connection
*    ** to selected ADO/ODBC database
*    ---------------------------------
     CALL PROCEDURE 'SQLDBCLOSECONN'
          USING SQL_SOCKET RETURNING SQL_RTN.
```

SQLBeginTran

This function is the COBOL version of the SQL_BeginTran procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Begin SQL Server Transaction
*    ---------------------------------
     CALL PROCEDURE 'SQLBEGINTRAN'
          USING SQL_SOCKET RETURNING SQL_RTN.
```

SQLCommitTran

This function is the COBOL version of the SQL_CommitTran procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Commit SQL Server Transaction
*    ---------------------------------
     CALL PROCEDURE 'SQLCOMMITTRAN'
          USING SQL_SOCKET RETURNING SQL_RTN.
```

SQLRollback

This function is the COBOL version of the SQL_Rollback procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Rollback changes made before COMMIT
*    ---------------------------------
     CALL PROCEDURE 'SQLROLLBACK'
          USING SQL_SOCKET RETURNING SQL_RTN.
```

SQLRunSQLSel

This function is the COBOL version of the SQL_RunSQLSel procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Build SQL Record Select String
*    ---------------------------------
     MOVE 'SELECT * FROM NameAddress' to SQL_SQLSEL.

*    ---------------------------------
*    ** Run SQL Query to Open ADO Recordset
*    ---------------------------------
     CALL PROCEDURE 'SQLRUNSQLSEL'
          USING SQL_SOCKET SQL_SQLSEL RETURNING SQL_RTN.
```

SQLRunSQLExec

This function is the COBOL version of the SQL_RunSQLExec procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Build SQL record insert string
*    ---------------------------------
     MOVE 'INSERT INTO NameAddress(First,Last,Address1,Address2,Ci
-    'ty,State,Zip,Phone,Fax,Email,Date1,Dollars)  Values('James',
-    ''Johnson','111 Main Street','Apt 5','Mpls','MN','55555','111
-    '-111-1111','222-222-2222','james@johnson.com','12/25/2002',
-    '123456.78)'
     to SQL_SQLEXEC

*    ---------------------------------
*    ** Run SQL Query to Insert record to NameAddress
*    ** NameAddress Access Table
*    ---------------------------------
     CALL PROCEDURE 'SQLRUNSQLEXEC'
          USING SQL_SOCKET SQL_SQLEXEC RETURNING SQL_RTN
```

SQLMoveFirst

This function is the COBOL version of the SQL_MoveFirst procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Move to first record
*    ---------------------------------
     CALL PROCEDURE 'SQLMOVEFIRST'
          USING SQL_SOCKET RETURNING SQL_RTN
```

SQLMoveLast

This function is the COBOL version of the SQL_MoveLast procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Move to last record
*    ---------------------------------
     CALL PROCEDURE 'SQLMOVELAST'
          USING SQL_SOCKET RETURNING SQL_RTN
```

SQLMoveNext

This function is the COBOL version of the SQL_MoveNext procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Move to next record
*    ---------------------------------
     CALL PROCEDURE 'SQLMOVENEXT'
          USING SQL_SOCKET RETURNING SQL_RTN
```

SQLMovePrev

This function is the COBOL version of the SQL_MovePrev procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Move to previous record
*    ---------------------------------
     CALL PROCEDURE 'SQLMOVEPREV'
          USING SQL_SOCKET RETURNING SQL_RTN
```

SQLGetField

This function is the COBOL version of the SQL_GetField procedure.

SQLGetFldChr

This function is the COBOL version of the SQL_GetFldChr procedure.

COBOL example usage:

```
*    ---------------------------------
*    ** Get value for selected field
*    ** in recordset. Return to 256
*    ** character work field.
*    ---------------------------------
     MOVE 'lastname' to SQL_FLDNAME.

     CALL PROCEDURE 'SQLGETFLDCHR'
          USING SQL_RECORD SQL_FLDNAME RETURNING WORKCHAR
```

SQLGetFldNum

This function is the COBOL version of the SQL_GetFldNum procedure.

SQLGetFldDat

This function is the COBOL version of the SQL_GetFldDat procedure.

SQLMoveFirsBuf

This function is the COBOL version of the SQL_MoveFirstBuf procedure.

SQLMoveLastBuf

This function is the COBOL version of the SQL_MoveLastBuf procedure.

SQLMoveNextBuf

This function is the COBOL version of the SQL_MoveNextBuf procedure.

SQLMovePrevBuf

This function is the COBOL version of the SQL_MovePrevBuf procedure.

SQLGetFldChrB

This function is the COBOL version of the SQL_GetFldChrB procedure.

SQLGetFldNumB

This function is the COBOL version of the SQL_GetFldNumB procedure.

SQLGetFldDatB

This function is the COBOL version of the SQL_GetFldDatB procedure.

SQLGetStrRow

This function is the COBOL version of the SQL_GetStrRow procedure.

SQLGetFldDat2B

This function is the COBOL version of the SQL_GetFldDat2B procedure.

SQLGetDelimRcd

This function is the COBOL version of the SQL_GetDelimRcd procedure.

SQLGetDelimFld

This function is the COBOL version of the SQL_GetDelimFld procedure.

SQLGetDelimFlt

This function is the COBOL version of the SQL_GetDelimFlt procedure.

SQLGetFldCount

This function is the COBOL version of the SQL_GetFldCount procedure.

SQLPCFileEmpty

This function is the COBOL version of the SQL_PCFileEmpty procedure.

SQLPCFileExist

This function is the COBOL version of the SQL_PCFileExist procedure.

SQLPCFileRen

This function is the COBOL version of the SQL_PCFileRen procedure.

SQLPCFileCopy

This function is the COBOL version of the SQL_PCFileCopy procedure.

SQLPCFileDel

This function is the COBOL version of the SQL_PCFileDel procedure.

SQLLastErrNum

This function is the COBOL version of the SQL_LasrErrNum procedure.

SQLLastErrMsg

This function is the COBOL version of the SQL_LasrErrMsg procedure.

SQLLastFullErr

This function is the COBOL version of the SQL_LasrFullErr procedure.

SQLConcat

The **SQLConcat** API allows COBOL users to quickly concatenate two variables together and return the resulting string.

Returns
Text string value containing concatenated values.

SQLConcatFld

The **SQLConcatFld** API allows COBOL users to quickly concatenate a field variable together with field delimiters and record delimiters for use when generating a SQL INSERT, UPDATE or DELETE string to add a record to an SQL database.

Returns
Text string value containing concatenated values.

Appendix B: List of Data Areas

This appendix lists the RPG2SQL Integrator data areas. You can use these data areas to control functions. Data areas are in the RJSRPGSQL library.

RJS Software recommends that you avoid modifying data areas that are not in the list below.

Use these AS/400 commands to modify the data areas:

- WRKDTAARA - The Work with Data Areas command allows you to show a list of data areas from one or more libraries.
- CHGDTAARA - The Change Data Area command allows you to change a data area.

These are some of the values contained in the RPG2SQL Integrator data areas that you may need to modify:

Data Area	Description
LPAR	Stores the LPAR Access Codes. *Value Type:* This is a character value with a length of 1. A sample entry would be '1'.
RJSRPGSQL	Stores the RPG2SQL Integrator product-authorization (license) key. The authorization code can be set by using the PRDSEC command. *Possible Values:* License key provided by RJS Software. This is a character value with a length of 50.
SQLASCII	Translates from ASCII to EBCDIC in your national language. *Possible Value:* This is a character value with a length of 10 *Default Value:* 'Q037337850'
SQLEBCDIC	Translates from EBCDIC to ASCII in your national language. *Possible Value:* This is a character value with a length of 10 *Default Value:* 'Q850337037'
VERSION	Contains the current PRG2SQL Integrator library-version information. RJS Software support staff can use the software version to diagnose problems. *Value Type:* This is a character value with a length of 50. A valid entry would be 'RPG2SQL Integrator Version V1.50'.

Appendix C: AS/400 Commands

Appendix C: AS/400 Commands

This appendix contains detailed descriptions of and command syntax for each RPG2SQL Integrator AS/400 command.

Check for IFS File Existence (CHKOBJIFS)

Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Check for IFS File Existence (CHKOBJIFS) command checks for the existence of an IFS directory or file. The command returns a CPF9898 message if the IFS file is not found. The command returns a CPF9897 message if the IFS file is found.
Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
FILNAM	IFS file name	*Character value*	Required, Positional 1

Top

IFS file name (FILNAM)
Specifies the IFS file or directory whose existence you want to check.
This is a required parameter.
character-value
Specify the name of the IFS file path. This path can be absolute or relative.

Top

Example for CHKOBJIFS
Note: This example assumes you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.

```
CHKOBJIFS FILNAM('/RJSRPGSQL/INVOICE.PDF')
```

This command checks for the existence of the file /RJSRPGSQL/INVOICE.PDF.

Top

Error messages for CHKOBJIFS
***ESCAPE Messages**
CPF9898
&1.

Top

Display OS/400 Level (PRDINFO)

Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Display OS/400 Level (PRDINFO) command displays the following information about your System i:

- The operating-system level.
- The serial number.
- The model number.
- The logical-partition (LPAR) number.

Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- This command is not threadsafe.

There are no parameters for this command.

Top

Parameters
None

Top

Example for PRDINFO
Note: This example assumes you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.

```
PRDINFO
```

This command displays the following information about your System i on the status line of your 5250 session:

- The operating-system level.
- The serial number.
- The model number.
- The LPAR number.

For example: OS/400 Level: 530 Model: 520 Serial#: 106E02E LPAR: 1

Top

Error messages for PRDINFO
***ESCAPE Messages**
CPF9898
&1.

Top

Enter Access Codes (PRDSEC)
Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Enter Access Codes (PRDSEC) command enters access codes for RPG2SQL Integrator.
Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- You must have the license key that RJS Software provides upon purchasing RPG2SQL Integrator.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
SECURITY	Enter security access code	*Character value*	Required, Positional 1
DTAARA	Enter data area name	*Character value*	Required, Positional 2
DTALIB	Enter data area library	*Character value*	Required,

	name		Positional 3

Top

Enter security access code (SECURITY)
Specifies the 50-character security code that RJS Software provides.
This is a required parameter.
character-value
Specify the security code.

Top

Enter data area name (DTAARA)
Specifies the data area that contains the security code. RJS Software provides this information with the license key.
This is a required parameter.
character-value
Specify the name of the data area.

Top

Enter data area library name (DTALIB)
Specifies the data-area library. RJS Software provides this information with the license key.
This is a required parameter.
character-value
Specify the data-area library name.

Top

Example for PRDSEC
Note: This example assumes you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.

```
PRDSEC SECURITY('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
       DTAARA(RJSRPGSQL)
       DTALIB(RJSRPGSQL)
```

This command enters a 50-character security-access code in the data area RJSRPGSQL in the data-area library RJSRPGSQL.

Top

Error messages for PRDSEC
***ESCAPE Messages**
CPF9898
&1.

Top

Test ADO Connection String (RSQCONN)

Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Test ADO Connection String (RSQCONN) command tests connection strings, tests if the server is running, and tests if you can connect successfully. A successful connection displays a message that tells you the database connection was successful. If the connection fails, you will recieve an error message. This command is an example of how to use our error-handling functions.
Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
HOST	RPG2SQL Server IP Address	*Character value*	Required, Positional 1
ADODSN	ADO Data Source String/Login	*Character value*, **'Driver;Server;Database;UID;PWD;'**	Optional, Positional 2

Top

RPG2SQL Server IP Address (HOST)

Specifies the IP address or host name where the RPG2SQL Integrator PC component is running. This is a required parameter.
character-value
Specify the IP address or host name.

Top

ADO Data Source String/Login (ADODSN)

Specifies the OLE DB-ODBC connection string that you want to test.
'Driver;Server;Database;UID;PWD;'
This entry is a skeleton, or a template, of an SQL Server-ODBC connection string.
character-value
Specify the connection string.
Note: As of the printing of this manual, two websites where you can find information on building connection strings for many database management systems are:

- http://www.connectionstrings.com.
- http://www.carlprothman.net then click 'connection strings'.

Top

Examples for RSQCONN
Note: These examples assume you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.
Example 1: Connecting to Microsoft Access

```
RSQCONN  HOST('125.1.2.49') ADODSN('Driver={Microsoft Access Driver (*.mdb)};
Dbq=c:\program files\rpgsqlsv\irpgsql.mdb; Uid=admin; Pwd=;')
```

This command connects to Microsoft Access.
Example 2: Connecting to SQL Server

```
RSQCONN  HOST('125.1.2.49') ADODSN('Driver={Microsoft Access Driver (*.mdb)};
Dbq=c:\program files\rpgsqlsv\irpgsql.mdb; Uid=admin; Pwd=;')
```

This command connects to SQL Server.

Top

Error messages for RSQCONN
***ESCAPE Messages**
CPF9898
&1.

Top

Return device IP address (RSQIP)

Where allowed to run:

Parameters
Examples
Error messages

- Batch program (*BPGM)
- Interactive program (*IPGM)

Threadsafe: No

The Return device IP address (RSQIP) command returns the TCP/IP address for the current 5250 session.
Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- You must call this command from a CL program.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
DEVD	Device	*Name*, *	Optional, Positional 1
IPADDR	IP address (15)	*Character value*	Optional, Positional 2

Top

Device (DEVD)
Specifies the System i display-device name.
*
Use this entry to retrieve the IP address of the the current 5250 session.
name
Specify the name of the System i display-device name.

Top

IP address (15) (IPADDR)
This parameter returns a 15-character IP address for the selected device or current 5250 session.

Top

Example for RSQIP
Note: This example assumes you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.

```
RSQIP   DEVD(*)
```

This command returns the 15-character IP address for the current 5250 session.

Top

Error messages for RSQIP
***ESCAPE Messages**
CPF9898
&1.

Top

Ping Remote Host (SQLPING)

Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Ping Remote Host (SQLPING) command tests the connectivity between the System i and the remote system specified by the *Remote System (RMTSYS)* parameter.
Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
RMTSYS	Remote System	*Character value*, *INTNETADR	Required, Positional 1
INTNETADR	Remote internet address	*Character value*	Optional, Positional 2

Top

Remote System (RMTSYS)
Specifies the remote-system name of the host with which the PING operation takes place. To be successful, the name must be valid, and the remote system must be able to communicate with the local system. You can assign names to an internet address by using either of the following:

- Work with Host Table menu, which is an option on the Configure TCP/IP (CFGTCP) menu.
- Remote name server to map a remote system name to an internet address.

This is a required parameter.
***INTNETADR**
The remote system is identified by the value specified for the *Remote internet address (INTNETADR)* parameter.
character-value
Specify the remote system name to be verified.

Top

Remote internet address (INTNETADR)
Specifies the remote internet address. Either a valid IP Version 4 or IP Version 6 address will be accepted. An IP Version 4 internet address is not valid if it has a value of all binary ones or all binary zeros for the network identifier (ID) portion or the host ID portion of the address.
character-value
Specify the internet address of the remote system. If you enter the internet address from a command line, you must enclose the address in apostrophes.

Top

Example for SQLPING
Note: This example assumes you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.

```
SQLPING  RMTSYS('www.ibm.com')
```

This command tests if you can connect to the specified address, www.ibm.com.

Top

Error messages for SQLPING
***ESCAPE Messages**
CPF9898
&1.

Top

Import Worksheet to PF (SQLXLSIMP)

Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Import Worksheet to PF (SQLXLSIMP) command imports a Microsoft Excel worksheet to a physical file member. Data is imported according to the definitions of the fields in the physical file. Warnings can be issued for invalid data.
Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- The physical file must already exist.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
INXLS	From workbook	*Path name*	Required, Positional 1
INSHEET	From worksheet	*Character value*, ***ACTIVE**	Optional, Positional 8

OUTFILE	To file	*Qualified object name*	Required, Positional 2
	Qualifier 1: To file	*Name*	
	Qualifier 2: Library	*Name*, ***LIBL**, *CURLIB	
OUTMBR	To member options	*Element list*	Optional, Positional 3
	Element 1: Member	*Name*, ***FIRST**	
	Element 2: Replace or add records	***REPLACE**, *ADD	
HDRROWS	Header rows	*Decimal number*, ***NONE**	Optional, Positional 4
LOGMSG	Log messages	***YES**, *NO	Optional, Positional 5
HOST	RPG2SQL Integrator host name	*Character value*, ***CURRENT**	Optional, Positional 6
BLKROW	Blank row option	***SKIP**, *DFT	Optional, Positional 7

Top

From workbook (INXLS)
Specifies the file path of the workbook that you want to import from the PC where the RPG2SQL Integrator PC component is running.
This is a required parameter.
path-name
Specify the path name of the workbook.

Top

From worksheet (INSHEET)
Specifies the name of the worksheet from which the data will be imported.
***ACTIVE**
Specifies the worksheet that was active when you saved the workbook.
character-value
Specify the name of the worksheet.

Top

To file (OUTFILE)
Specifies the physical file to which to import the worksheet.
This is a required parameter.
Qualifier 1: To file
name
Specify the name of the physical file.
Qualifier 2: Library
***LIBL**
Searches all libraries in the current library list until the first match is found.
***CURLIB**
Specifies the current library.
name

Specify the name of the library.

Top

Output member options (OUTMBR)
Specifies the name of the database file member to which the output is directed.
Element 1:
***FIRST**
The first member of the specified file is used.
name
Specify the name of the file member. If the member does not exist, it will be created.
Element 2: Replace or add records
***REPLACE**
The file is cleared before new records are inserted.
***ADD**
New records are added after any existing records.

Top

Header rows (HDRROWS)
Specifies the number of rows that comprise the header. These rows are not included in the data that is imported.
***NONE**
None of the rows comprise the header.
decimal-number
Specify the number of rows that comprise the header.

Top

Log messages (LOGMSG)
Specifies whether to log warnings and errors that occur when importing the worksheet to the physical file.
***YES**
Log warnings and errors.
***NO**
Do not log warnings or errors.

Top

RPG2SQL Integrator host name (HOST)
Specifies the IP address of the PC on which the RPG2SQL Integrator PC component is running.
***CURRENT**
Specifies to use the IP address of your 5250 session.
character-value
Specify the IP address of the PC on which the RPG2SQL Integrator PC component is running.

Top

Blank row option (BLKROW)
Specifies whether to import blank rows from a worksheet.
***SKIP**
Do not import blank rows.
***DFT**

Import blank rows using default values. Numeric fields are set to 0. All other fields are set to blanks.

Top

Examples for SQLXLSIMP
Note: These examples assume you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.
Example 1: Simple Command Example

```
SQLXLSIMP INXLS('C:\EXCEL FILES\SAMPLE.XLS')
          OUTFILE(RJSLIBRARY/SAMPLEPF)
```

This command imports the active worksheet of workbook C:\Excel Files\Sample.xls to the first member of physical file SAMPLEPF in library RJSLIBRARY, using the following default settings:

- Use the first member of SAMPLEPF.
- Replace existing data.
- There are no header rows.
- Log warnings and errors.
- Use the IP address of your 5250 session.
- Skip blank rows.

Example 2: More Complex Command Example

```
SQLXLSIMP INXLS('C:\EXCEL FILES\SAMPLE.XLS')
          OUTFILE(RJSLIBRARY/SAMPLEPF)
          OUTMBR(QUARTER2 *ADD)
          HDRROWS(3)
          LOGMSG(*YES)
          HOST(1.1.1.1)
          BLKROW(*DFT)
```

This command imports the active worksheet of workbook C:\Excel Files\Sample.xls to member QUARTER2 of physical file SAMPLEPF in library RJSLIBRARY, using the following settings:

- Use member QUARTER2.
- Add new records after any existing records.
- There are three header rows.
- Log warnings and errors.
- Use IP address 1.1.1.1.
- Use default values for blank rows.

Top

Error messages for SQLXLSIMP
***ESCAPE Messages**
RPS9995
Warnings issued while importing worksheet &1(&2) - DSPJOB &5/&4/&3 for diagnostic messages.
RPS9999
Error occured while importing worksheet &1(&2) - DSPJOB &5/&4/&3 for diagnostic messages.

Top

Export PF to Worksheet (SQLXLSEXP)

Note: As of the printing of this guide, this command, SQLXLSEXP, is under development. It is not available for RPG2SQL users.
Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Export PF to Worksheet (SQLXLSEXP) command exports a physical file member to a Microsoft Excel worksheet.
Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
INFILE	From file	*Qualified object name*	Required, Positional 1
	Qualifier 1: From file	*Name*	
	Qualifier 2: Library	*Name*, ***LIBL**, *CURLIB	
INMBR	From member	*Name*, ***FIRST**	Optional, Positional 3
OUTXLS	To workbook	*Path name*	Required, Positional 2
OUTSHEET	To worksheet	*Character value*, ***ACTIVE**	Optional, Positional 4
HEADER	Header	***COLHDG**, *TEXT, *NAME, *NONE	Optional, Positional 5
LOGMSG	Log messages	***YES**, *NO	Optional, Positional 6
HOST	RPG2SQL Integrator host name	*Character value*, ***CURRENT**	Optional, Positional 7

RCDFIRST	First record to export	*Integer*, ***FIRST**	Optional, Positional 8
RCDLAST	Last record to export	*Integer*, ***LAST**	Optional, Positional 9
FLDFIRST	First field to export	1-8000, ***FIRST**	Optional, Positional 10
FLDLAST	Last field to export	1-8000, ***LAST**	Optional, Positional 11

Top

From file (INFILE)
Specifies the physical file on the System i whose member is to be exported to the worksheet.
This is a required parameter.
Qualifier 1: From file
name
Specify the name of the physical file.
Qualifier 2: Library
***LIBL**
Searches all libraries in the current library list until the first match is found.
***CURLIB**
Specifies the current library.
name
Specify the name of the library.

Top

From member (INMBR)
Specifies the name of the physical file member that is to be exported to the worksheet.
***FIRST**
The first member of the specified file is used.
name
Specify the name of the file member.

Top

To workbook (OUTXLS)
Specifies the file path of the workbook to which the physical file member will be exported. This file path must be available to the PC where the RPG2SQL Integrator PC component is running. It can also be located on a network file share.
This is a required parameter.
path-name
Specify the path name of the workbook.

Top

To worksheet (OUTSHEET)
Specifies the name of the worksheet to which the data will be exported.
***ACTIVE**
Specifies the worksheet that was active when you saved the workbook.
character-value

Specify the name of the worksheet. If the worksheet you specify does not exist, it will be created, if possible.

Top

Header (HEADER)
Specifies the data that appears in the worksheet's header row.
***COLHDG**
Specifies that the header text comes from the column headings of the fields of the physical file. If the column heading for a field does not exist, the text description will be used. If the text description does not exist, the field name will be used.
***TEXT**
Specifies that the header text comes from the text descriptions of the fields of the physical file. If the text description for a field does not exist, the field name will be used.
***NAME**
Specifies that the header text comes from the names of the fields of the physical file.
***NONE**
Specifies that there is no header.

Top

Log messages (LOGMSG)
Specifies whether to log warnings and errors that occur when exporting the physical file to the worksheet.
***YES**
Log warnings and errors.
***NO**
Do not log warnings or errors.

Top

RPG2SQL Integrator host name (HOST)
Specifies the IP address of the PC on which the RPG2SQL Integrator PC component is running.
***CURRENT**
Specifies to use the IP address of your 5250 session.
character-value
Specify the IP address of the PC on which the RPG2SQL Integrator PC component is running.

Top

First record to export (RCDFIRST)
Specifies the Relative Record Number (RRN) of the first record of the physical file member to be exported to the worksheet.
***FIRST**
The first record of the member is the first to be exported to the worksheet.
integer
Specify the RRN of the first record to be exported to the worksheet.

Top

Last record to export (RCDLAST)
Specifies the RRN of the last record of the physical file member to be exported to the worksheet.
***LAST**
The last record of the member is the last to be exported to the worksheet.

integer
Specify the RRN of the last record to be exported to the worksheet.

Top

First field to export (FLDFIRST)
Specifies the number of the first field of each record of the physical file member to be exported to the worksheet.
***FIRST**
The first field of each record is the first to be exported to the worksheet.
1-8000
Specify the number, from 1-8000, of the first field to be exported to the worksheet.

Top

Last field to export (FLDLAST)
Specifies the number of the last field of each record of the physical file member to be exported to the worksheet.
***LAST**
The last field of each record is the last to be exported to the worksheet.
1-8000
Specify the number, from 1-8000, of the last field to be exported to the worksheet.

Top

Examples for SQLXLSEXP
Note: These examples assume you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.
Example 1: Simple Command Example

```
SQLXLSEXP INFILE(RJSLIBRARY/SAMPLEPF)
          OUTXLS('C:\EXCEL FILES\SAMPLE.XLS')
```

This command exports the first member of physical file SAMPLEPF in library RJSLIBRARY to the active worksheet of workbook C:\Excel Files\Sample.xls, using the following default settings:

- Use the first member of SAMPLEPF.
- Use column headings for the header text.
- Log warnings and errors.
- Use the IP address of your 5250 session.
- Use blanks if the fields are null.
- Use the first record as the first one to be exported.
- Use the last record as the last one to be exported.
- Use the first field as the first one to be exported.
- Use the last field as the last one to be exported.

Example 2: More Complex Command Example

```
SQLXLSEXP INFILE(SAMPLEPF/RJSLIBRARY)
```

```
INMBR(QUARTER2)
OUTXLS('C:\EXCEL FILES\SAMPLE.XLS')
OUTSHEET('Sheet2')
HEADER(*NAME)
LOGMSG(*NO)
HOST(1.1.1.1)
NULLOPT(*DFT)
RCDFIRST(6)
RCDLAST(15)
FLDFIRST(4)
FLDLAST(12)
```

This command exports member QUARTER2 of physical file SAMPLEPF in library RJSLIBRARY to worksheet Sheet2 of workbook C:\Excel Files\Sample.xls, using the following settings:

- Use member QUARTER2.
- Use field names for the header text.
- Do not log warnings and errors.
- Use IP address 1.1.1.1.
- Use default values for fields that are null.
- Use the sixth record as the first one to be exported.
- Use the 15th record as the last one to be exported.
- Use the fourth field as the first one to be exported.
- Use the 12th field as the last one to be exported.

Top

Error messages for SQLXLSEXP

***ESCAPE Messages**

RPS9986

Warnings issued while exporting file &1/&2/&3 - DSPJOB &6/&5/&4 for diagnostic messages.

RPS9989

Error occurred while exporting file &1/&2/&3 - DSPJOB &6/&5/&4 for diagnostic messages.

Top

Upgrade Settings (SQLUPG)

Where allowed to run: All environments (*ALL)
Threadsafe: No

Parameters
Examples
Error messages

The Upgrade Settings (SQLUPG) command upgrades RPG2SQL Integrator software. Run this command to copy settings from an old library after upgrading to a new version of the RJSRPGSQL library.

Restrictions:

- You must have RJSRPGSQL added to your library list or qualify the command with the library name.
- This command is not threadsafe.

Top

Parameters

Keyword	Description	Choices	Notes
OLDLIB	Old library	*Character value*	Required, Positional 1
NEWLIB	New library	*Character value*, **RJSRPGSQL**	Optional, Positional 2

Top

Old library (OLDLIB)
Specifies the re-named version of the RJSRPGSQL library.
This is a required parameter.
character-value
Specify the the re-named library.
For example: RJSRPGSQLV

Top

New library (NEWLIB)
Specifies the name of the new or upgraded RJSRPGSQL library
RJSRPGSQL
Specifies the library name RJSRPGSQL.
character-value
Specify the the new or upgraded libarary name.

Top

Example for SQLUPG
Note: This example assumes you have added RJSRPGSQL to your library list. Otherwise you must qualify the command with the library RJSRPGSQL.

```
SQLUPG OLDLIB(RJSRPGSQLV)
```

This command copies settings from library RJSRPGSQLV to library RJSRPGSQL.

Top

Error messages for SQLUPG
***ESCAPE Messages**
CPF9898
&1.

Top