sdOffice™

Automate Office From Anywhere

Version 1.0

Table of Contents

Overview

sdOffice™ is a Windows-based product that enables automation of Microsoft Office programs from software environments that don't support Microsoft's ActiveX Automation standard. Through the use of platform-independent TCP/IP sockets, or the older, more widely supported Windows DDE standard, sdOffice enables advanced automation tasks to be developed in most any language, and executed on most any platform.

For example, a customer maintenance program, running in a terminal emulator window, on a Linux system could start Microsoft Word on the user's (or another user's) workstation, open a letter-head template, write a letter incorporating the customer address and balance information, and print, fax, or email the letter.

Another example is the need to export data to a file and import it into Microsoft Excel. This multi-step process results in an unformatted worksheet. With sdOffice, it is possible for an external program to launch Excel, create a new worksheet, add the data and/or formulas, add formatting, and even sort or subtotal the data.

More examples might be to synchronize a user's Outlook Contacts database with an accounting system master file, or post a reminder to run a report at a future date in his or her Outlook Calendar.

By providing a concise command-style interface, sdOffice is very easy to learn. Common tasks are simple to develop and execute.

sdOffice Provides Multiple Programming Interfaces

Socket Interface

For some host environments, such as Unix or Linux, sdOffice provides a TCP/IP server that runs on a Windows workstation, accepting automation commands over a socket from any TCP/IP-enabled program. Automation tasks can be written in Java, Perl, Tcl, or even Telnet.

When using the socket interface, sdOffice is started as a TCP/IP server on the user's workstation. It listens for connections from another (or the same) system, and provides a simple command-response interface that is easy to program and even easy to use via a telnet session.

DDE Interface

Some Windows programming languages are able to communicate with other Windows tasks via DDE, but not ActiveX automation. sdOffice provides a DDE interface for those languages.

BBx and ProvideX Interface

The initial vision for sdOffice was as a set of Visual PRO/5 and ProvideX programs designed to manage Microsoft Office via DDE. These programs have been enhanced to support both DDE and socket communication with sdOffice, providing and easy CALL interface for automation tasks.

In addition to the above low-level interfaces, you can use one of the sdRun interfaces, which accept a command file as input and manage the server communication automatically.

Installation and Licensing

Installation
sdOffice is installed on each workstation by running the setup.exe program from the CD's sdoffice/win directory, or by running the self-installing executable downloaded from SDSI's website. The installation is controlled by InstallShield, an industry-standard installation utility.

Once installed, there will be a Startup menu launch of the server when the user logs into the workstation, and also a manual startup available from the desktop or the Start menu. sdOffice can be configured to only be started manually, and can be hidden automatically when started from the Startup menu. When hidden, you can restore or shutdown the server from the system's task tray by clicking the small sdOffice icon.

In the CD's sdoffice/unix directory, and also included in the unix.tar file in the sdOffice installation directory, are several files that are of benefit to Unix or Linux developers. The tar archive can be copied to a Unix system directory and installed with the command **tar xvf unix.tar**. The files include the perl script sdrun.pl, the sample command files s_*.txt, and documentation in PDF format.

Licensing
If sdOffice will accept connections via TCP/IP sockets, then any system on your network can control an sdOffice session on any workstation. It is possible, in cases where no user interaction is required, to develop automations using sdOffice on one workstation to service the entire network. sdOffice can therefore be licensed one of two ways: as a workstation or as a server.

License keys are character strings that are entered into the sdOffice configuration dialog. The license key is based upon the computer's system ID, the version of sdOffice, and the mode of operation. License keys are purchased from the dealer or publisher. Upon purchase, you will receive an order number and PIN code that can be used to obtain one or more license keys from website http://synergetic-data.com. If the workstation has access to the Internet, then the Get License button on the configuration dialog can be used to access the proper web page.

If a system has been licensed and then the system ID changes, sdOffice will run an additional 10 days in workstation mode before reverting to demo mode. Contact the dealer or publisher to obtain a new license key during that period.

Demo Mode
After installation, sdOffice operates in demo mode. In this mode, random letters are replaced with asterisk (*) characters when reading and writing data. Once a valid license key is entered into the configuration, then sdOffice starts operating in one of the two license modes. Demo mode can emulate either Workstation mode or Server mode, so testing of either license model is possible. If the license key is set to "demo", then sdOffice will operate in workstation mode. For server mode demo operation, set the license key to "demoserver".

Workstation Mode
A workstation license will accept any number of local connections, either DDE or socket connections to address 127.0.0.1. In addition, it will accept one external socket connection at a time.

Server Mode
A server license, like the workstation license, will accept any number of local connections, and will also support any number of simultaneous external connections.

Steps to License sdOffice

The first step to licensing sdOffice is to order a license or license pack from the dealer or publisher. You will receive an order information sheet by email, fax, or standard mail, containing an *order number* and *pin number*, which are used to get a license key from the Internet using a web browser such as Microsoft Internet Explorer or Netscape Navigator. The order information sheet contains detailed instructions, but the basic steps are as follows:

Use the sdOffice Configure button to view the configuration window. This window displays the system ID.

The order number, pin number, and system ID are used together to obtain a license key via a web browser. The website is http://synergetic-data.com; navigate to the sdOffice page, then the Licensing page. If the sdOffice machine can connect to the Internet, then you can use cut (Control-C) and paste (Control-V) keystrokes to copy the system ID to the browser, and the license key back to the configuration window. You can also launch your browser automatically with the system ID information pre-filled out using the Get License button on the configuration window.

Socket Interface

When sdOffice is installed, it is set up as a TCP/IP server on the workstation's Startup menu, so it is automatically launched when the system is started. It listens on a port, by default 6114, for connections from client tasks. Once a connection is made, sdOffice responds with the string "sdOffice (tm) <version> - from <client address>", and a conversation begins. The client sends commands followed by LF or CR-LF characters (LF=ASCII 10, CR=ASCII 13). sdOffice performs the requested command and returns a response. The response is either OK<CR-LF> or Error: *msg*<CR-LF>.

Upon opening the socket, the client should read the socket until a line starting with "sdOffice" is encountered. This will normally be the first line, and will clear the socket input buffer in preparation for the conversation.

The first command sent by the client needs to be an application invocation command: Excel, Word, Outlook, MAPI, or ADO (case-insensitive). Any other command will receive an error response. This command will start the appropriate application on the sdOffice system and await further commands. Once all commands have been entered and you are ready to close the session, enter the command Quit<CR-LF> to close the socket.

If a command requires parameters, then follow it with a space and the parameters. In some cases, multiple parameters are accepted or required. These are entered as a series of comma-separated options, each a *name* or *name=value* pair. If a *value* contains a comma, it must be quoted.

Some commands may require long parameter strings. If the entry or programming of such long strings is difficult, you can use the end-of-line continuation character, a backslash (\). sdOffice interprets a trailing \ character to indicate that more command data is coming on the next line. For example, this series of commands is interpreted as a single "write" command (note the use of a space before each \, to prevent words from combining together):

```
write This is the start of a paragraph, \
this is still more paragraphs text, \
and this is end of the  line.
```

In addition, parameter may require embedded CR-LF data. You can use the character sequence \n to indicate a CR-LF sequence, and the character sequence \t for tabs (ASCII 9). For example, a Word bullet list is added with embedded CR-LF sequences:

```
bulletlist Item number 1\nItem number 2\nItem number 3
```

Here is a simple example using telnet to demonstrate the conversation:

```
$ telnet 1.1.1.1 5000
Trying 1.1.1.1...
Connected to nt1.
Escape character is '^]'.
sdOffice (tm) - from 1.1.1.2
word
OK
newdoc
OK
font name=Arial,size=14,bold
OK
write Now is the time for all good men to come to the aid
of their country.
OK
print
OK
quit
Connection closed by foreign host.
```

Some commands cause sdOffice to return data. These commands all start with the letters "Get" The data returned may contain CR-LF sequences (the paragraphs of a Word document, for example), so all Get commands are followed by single line with just a period (.) to indicate the end of the data. When reading the response, be sure to read through a line containing a .<CR-LF> , discarding the final line, to ensure that all the data is read. Note that if you encounter a line with two periods (..<CR-LF>), this indicates that the line is not the end of the transmission, but in fact a line containing a single period.

One of the challenges of using sdOffice in a network environment, with a Unix host executing an application via telnet or other network protocol, is determining the user's workstation computer. The Unix operating system provides workstation host name information with the "who" command. There are two variants of who. Most provide hostname information inside parenthesis, while SCO adds the hostname as the sixth element of who output when the -x argument is added. Here are shell command lines that set the variable host correctly:

SCO: host=`who -mx | tr -s " " | cut -f6 -d" "`

Other: host=`who -m | sed -e 's/.*(//; s/)//'`

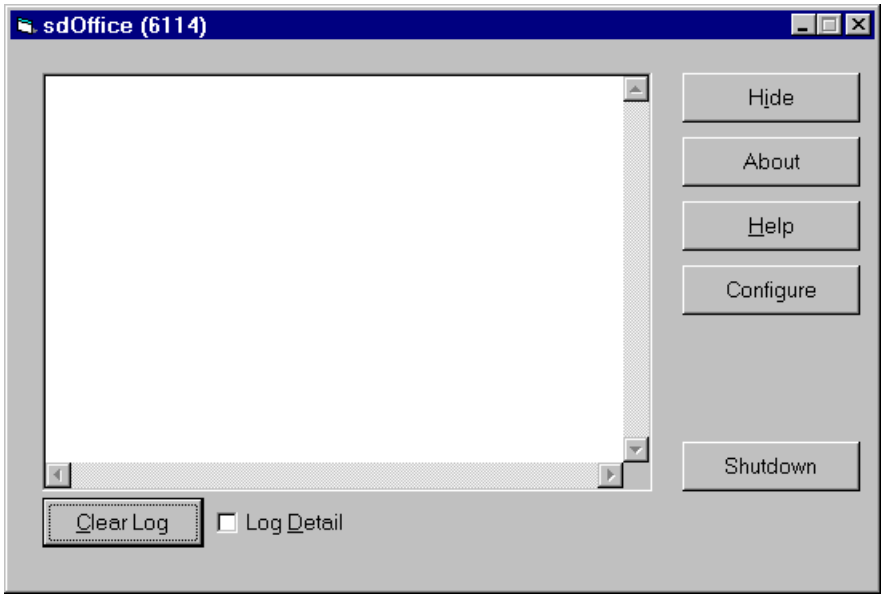
The Perl programs provided for socket communications, sdrun.pl and sdpipe.pl, both use the environment variable $SDHOST, if available, rather than the who command. This provides a method to communicate to a sdOffice server that is

not running on the user's workstation, and also helps in cases where the who command returns incorrect information.

TCP/IP server

The sdOffice server is launched from the user's desktop, and operates as a TCP/IP server application, listening on a configured port (default=6114) for connections from client applications. These clients converse with the server using sdOffice's socket interface. Connects and disconnects are displayed in the scrolling text log. Optionally, commands issued by clients can also be logged.

When started, the server will display the startup time, listening port, optional binding information, and the license mode (demo, workstation, or server). As connections are made, external connections (to other than the localhost 127.0.0.1 address) are counted and displayed in parentheses. Workstation licenses support one external connection at a time, while server licenses support any number of external connections.



Hide
The Hide button will hide the server without shutting it down. It will still listen for, and process, connections. To force the server to be visible again, use the system tray icon on the Windows task bar. Double-click the icon, or right click and choose Restore from the popup menu.

About
The About button displays version and copyright information about sdOffice.

Help
The Help button displays this help screen.

Configure
The Configure button displays the configuration dialog. If you change the configuration, you will be prompted to restart the server when you exit.

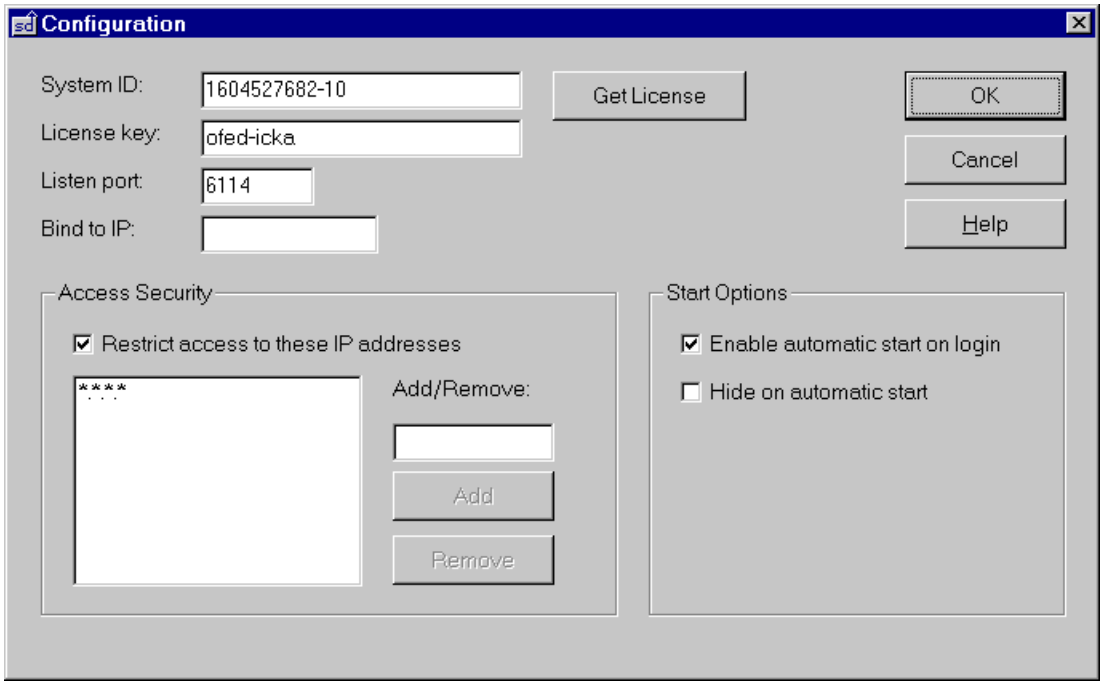
Shutdown
The Shutdown button, or closing the window, will stop the server, after a verification prompt. Shutting down the server will close any connections that are active without warning and without completing any pending commands.

Logging
The Clear Log button will clear the scrolling text log.

If the Log Detail check box is checked, then all commands are displayed in the log. Otherwise, only connects and disconnects are displayed. When detail is shown, any character codes outside the ANSI printable range (below ANSI 32) are displayed in angle brackets. <9> would represent a tab character, for example. The log is limited to 1000 lines; prior lines are removed as lines are added.

Server Configuration

The server is configured by clicking the Configure button while running the TCP/IP server, or by running the Configure sdOffice option from the Start menu. This displays the configuration dialog window, where you can specify information on server operation and licensing.



System ID
The System ID is used for licensing purposes. Each workstation has a unique ID derived from the Windows serial number and the hard disk serial number. This information can't be changed.

Get License Button
Once you have purchased a license or license pack, you will receive an order number and PIN code. With this information, and the system ID information shown, you can obtain a license key via a web browser. If the system running sdOffice is connected to the Internet, clicking this button will launch your browser, showing the sdOffice licensing web page with the system ID pre-entered.

License Key
The License key is linked to the system ID to enable normal operation of the server. Until a valid license key is entered, sdOffice will operate in demo mode. Demo mode causes random characters to be replaced with asterisks (*) when reading and writing data to the various automation tasks. License key values used for demo mode are "demo" and "demoserver". Any other key, if not valid for the system on which sdOffice is running, will cause a warning message to display and demo mode to be invoked.

Listen Port
The Listen port is the TCP/IP port on which sdOffice listens for connections. All TCP/IP servers are assigned a listening port. Ports below 1024 are reserved for well-known applications, such as http (the Web) and smtp (email). The default sdOffice port is 6114, but you can modify it to any available port from 1024 thorugh 65535. Client connections need to specify this port when

connecting to the server.

Bind to IP
Normally, sdOffice listens to any IP address recognized by the machine. Machines can be configured with aliasing to support multiple IP addresses. By entering an IP address here, you are instructing sdOffice to bind to that address, and ignore connection attempts to any other address the machine might have. If you leave this entry blank, then sdOffice will respond to any connection attempt to the correct listen port number, regardless of the IP address of the connection.

Access Security
You can restrict access to specific IP addresses or wildcards by checking the "Restrict access ..." checkbox. Then add and remove addresses to the associated list box. To add an address or a wildcard, fill in Add/Remove box with an address in the format n.n.n.n and click the Add button. To remove an entry, select it in the list box and click the Remove button.

Address wildcards can use a * character in place of a number. For example, "10.20.30.*" would allow all addresses in the block 10.20.30.1 through 10.20.30.255 to access the server.

Start Options
The sdOffice server is configured during installation to automatically start when the user logs into the workstation. This action can be disabled by unchecking the "Enable automatic start on login" checkbox. In addition, when sdOffice starts in this manner, you can indicate it should run hidden by checking the "Hide on automatic start" checkbox.

A desktop shortcut is also created during installation. Starting sdOffice with that shortcut will always make the server visible.

BBx and ProvideX Interface

sdOffice includes several BBx and ProvideX programs that can simplify the management of a sdOffice session. These programs are CALLed modules that initiate and manage the communication with sdOffice automatically.

There is one program module for each automation type. For example, sdofc_e.bb manages and Excel session under BBx; sdofc_w.pv manages a Word session under ProvideX. The BBx programs are compatible with BBx4, PRO/5 and Visual PRO/5 versions of BBx. Note that the Unix versions in both languages communicate with sdOffice exclusively via a socket. The Windows versions in both languages will communicate via local DDE, unless the global string "$sdhost" is defined or the environment variable SDHOST is defined, in which case a socket is used. Visual PRO/5 versions prior to 2.2 cannot use a socket, and are therefore only capable of communicating with sdOffice via local DDE.

Each sdOffice program uses a simple, three-value argument list, **call "sdofc_e.bb", cmd$, response$, errmsg$** for example. In each case, cmd$ is a command value and optional parameters, response$ returns any results from "get" style commands, and errmsg$ returns a null or an error message, if an error is encountered. Unlike the lower-level socket or DDE interfaces, there is no need to initially specify which application to automate (Word, Outlook, Excel, etc.), as the program CALLed selects and initializes the correct application automatically.

In many cases, the parameter portion can contain a number of *name=value* pairs. Each pair is delimited with a comma, and each *value* may be quoted if it contains commas. For example, to set a cell font in Excel, you would use a command like this:

call "sdofc_e.bb","format col=1,row=1,font=Arial,size=14.5,bold",response$,errmsg$

The first CALL to one of the sdOffice programs will open a channel to the sdOffice server in either DDE or socket mode, and issue the correct initialization command (word, excel, mapi, etc.). The type of channel is determined by the environment:

- In either language, if stbl("$sdhost") or gbl("$sdhost") or the environment variable SDHOST is defined, then a socket channel is opened to the identified host IP address or name. In PRO/5 or Visual PRO/5 2.2 or higher, socket device N0 is opened (it must be defined in your config.bbx file). In lower releases of PRO/5 or BBx4 on Unix, a pipe to "|perl sdpipe.pl" is opened.

Visual PRO/5 prior to revision 2.2 cannot use a socket.

- In either language on a Windows workstation, if the above values aren't set, a DDE channel is used, requiring that sdOffice be installed on the local workstation. The sdofc_*.bb/pv programs must reside in the same directory as the sdofc.exe program, as the path to the sdofc.exe executable is derived from the program name called.

- In PRO/5 on a Unix system, if the above values aren't set, a pipe is used to open a socket channel via sdpipe.pl, which will attempt to determine a workstation hostname via the who command.

- In ProvideX, if the above values aren't set, either on Unix or in a WindX session, a socket channel is opened. On Unix, the who command is used to determine the user's workstation hostname. Under WindX, the WindX workstation hostname is used., so sdOffice would need to be running on the WindX workstation.

If a socket channel is opened, it will attempt to resolve the local workstation address using WindX.utl or Unix who commands, unless you specify an address by setting the global string (STBL or GBL) "$sdhost", or the environment variable "SDHOST", to an IP address or domain name of a sdOffice workstation. Further, it will use port 6114, unless you specify a different port in the global string "$sdport".

The channel must remain open, or the session terminates, along with the ActiveX Automation session. When you are ready for the session to end, you can call the program with cmd$="close", and the session will terminate. As an alternative, a Business Basic BEGIN or END will also close channels and terminate the session.

The sdofc.bb/pv programs use timeout error handling when communicating with the server, both under socket and DDE modes. The default timeout value for any operation is 30 seconds. You can adjust this value by setting the STBL (GBL under pvx) value for $sdtim to the number of seconds desired. For example, trash$=stbl("$sdtim","60") would establish 60 seconds as the timeout value before an error is returned.

Usage note: on Windows, the default sdOffice directory is "C:\Program Files\SDSI\sdOffice". If you include this in your BBx or ProvideX search prefix, the space in "Program Files" acts as a path delimiter and therefore doesn't work. Be sure to use the MS-DOS path name for "Program Files", which is generally "PROGRA~1". To verify, use the MS-DOS dir command from the C:\ directory.

There are five sdOffice programs. Here is a summary of each:

call "sdofc_d.bb|pv",cmd$,response$,errmsg$
This program automates ADO (database) tasks through SQL commands. With this object, you can read and write data in external databases.

call "sdofc_e.bb|pv",cmd$,response$,errmsg$
This program automates Excel, providing read, write, and formatting capabilities. You can open existing workbooks, create new workbooks, add embedded charts, manage worksheets and their contents, and print or save the results.

call "sdofc_m.bb|pv",cmd$,response$,errmsg$
This program automates MAPI (email messaging) tasks. You can send email to any number of recipients. The mail can include attachments.

call "sdofc_o.bb|pv",cmd$,response$,errmsg$
This program automates Outlook, managing appointments, contacts, email, and tasks. Date-oriented information can be used to automate appointment and task records. Master file data can be synchronized with Outlook contacts, and email can be sent using the Outlook address book. Note that email can also be sent using MAPI automation.

call "sdofc_w.bb|pv",cmd$,response$,errmsg$
This program automates Word, providing read, write, and formatting capabilities. Use it to open or create Word documents, such as writing letters or performing mail merge functions from within your application.

Each of these programs internally calls sdofc.bb|pv. That program should never be called directly.

DDE Interface

In the local Windows environment, it is possible to communicate with sdOffice via Dynamic Data Exchange (DDE) rather than via the socket interface. For languages that don't support sockets, but do support DDE, or for Windows programmers who are more familiar with DDE and whose applications won't need to work across a network, the DDE interface may be preferable. In this mode, sdofc.exe is started with two arguments, and it then operates as a DDE server waiting for a client to connect.

To start sdofc.exe as a DDE server, you must launch it in the following format:

sdofc.exe *type id*

The first parameter is an automation type, which can be word, excel, outlook, mapi, or ado. The second parameter is a unique numeric ID code, typically set to a process ID by the executing task.

DDE applications are identified by an Application name and a Topic. When using sdOffice, the Application name is "sdOffice", the Topic is "sdofc_*id*", where *id* is the value given on the command line. By using a unique value for a topic, the user can potentially have several DDE sessions operating at the same time.

Once a DDE session is opened, the two applications can exchange data through Items, and the client application can request the server execute a command.

There are two items used by a sdOffice DDE server, "Dat" and "ErrMsg". Dat is used to provide parameters for commands and to receive responses, while ErrMsg is used to report errors back to the client. The command itself is sent via an Execute command. This differs from the socket interface, in which both the command and its parameters are sent together in a single transaction, separated by a space.

Here, therefore, is some pseudo-code of a DDE conversation:

```
id$=str(handle)   (some unique value)
run "sdofc.exe word "+id$
repeat until successful:
        ddeopen (1) "sdOffice|sdofc_"+id$

execute(1)"newdoc"  (no parameters needed)

writeitem(1, item="dat")"Dear Kiki,\n\nI think you're cute."  (parameters for next command)
execute(1)"write"

If execute results in an error:
        readitem(1,item="errmsg")errmsg$

If lowercase(command) starts with "get":
        readitem(1,item="dat")response$

Repeat commands until done.

close(1)
```

sdRun Interfaces

sdOffice comes with a series of programs designed to process command files, simplifying the communication with sdOffice by allowing a developer to create a text file containing commands and then process the commands automatically. While there are no programming type features like looping constructs, variables, or conditionals, in many cases these interfaces will satisfy the needs of a project and will save the programmer the time of developing the communication layer. And for Perl or Business Basic programmers, the source code can be useful for seeing how to manage the communications.

The programs are:

sdrun.bb PRO/5 CALLable program
sdrun.pv ProvideX CALLable program
sdrun.pl Perl script program
sdrun.exe Windows 32-bit executable

Examples of command files can be found in the sdOffice directory, names s_*.txt. A command file always starts with an application name, such as "word", "excel", or "mail". Following this line are any number of commands and associated parameters. Comments can be interspersed as lines starting with #.

To execute the PRO/5 or Providex programs

call "sdrun.bb|pv", CommandFile$, ServerIP$, ServerPort$, Response$, Errmsg$

CommandFile$ is the command file path name to process.

ServerIP$ and ServerPort$ identify the server name (by IP address or hostname) and listening port. The default values are taken from the environment, and in cases where DDE will be used, these values are ignored. See the Visual PRO/5 and ProvideX interface page for more details.

Response$ will contain all the reponses returned by Get style commands in the command file. Multiple responses will be delimited with ASCII 0 characters.

Errmsg$ will return any error message encountered. If a command encounters an error, the remainder of the command file isn't processed, and the error message is returned immediately.

To execute the Perl program

perl sdrun.pl *CommandFile* {*ServerIP* {*ServerPort*}} {>*ResponseFile*} {2>*ErrorFile*}

CommandFile, the first argument, is required, and is the path to the command file.

ServerIP and ServerPort are optional arguments to specify the server IP address or hostname, and listening port. If they are not supplied, then defaults are used, taken from the environment variable SDHOST, or who command, and the environment variable SDPORT or port 6114. The perl program, unlike the PRO/5 or ProvideX programs, always uses the socket interface.

If any get-type commands are issued, the responses are sent to STDOUT. You can redirect the responses to a file rather than the terminal using ">" redirection. If there are multiple get-type commands, a line "<< multiple response break >>" will appear between each result set.

If any errors are encountered, the error message is sent to STDERR and the job exits immediately. You can redirect this output with "2>". Normally, STDERR is routed to the terminal.

To execute sdRun.exe

sdrun *CommandFile* {*ServerIP* {*ServerPort* {*ResponseFile* {*ErrorFile* }}}}

CommandFile, the first argument, is required, and is the path to the command file.

ServerIP and ServerPort are optional arguments to specify the server IP address or hostname, and listening port. If they are not supplied, then the defaults are localhost and 6114, respectively.

If any get-type commands are issued, the responses are normally displayed in a message box. You can direct the responses to a file using the ResponseFile argument as a file path. If there are multiple get-style commands, their respective responses will be delmited with formfeed (ASCII 12) characters.

If any errors are encountered, normally an error message window is displayed. You can direct this output to a file with the ErrorFile argument as a file path. In either case, the job exits immediately

Note that the arguments are positional. To name a ResponseFile, for example, you must also name the ServerIP and ServerPort.

ADO Automation

Microsoft's Active Data Object protocol is a database access protocol that provides access to local and remote databases via ODBC and OLE DB providers. sdOffice provides a simple interface to ADO, providing access to database table structures and SQL commands. Using this object requires a knowledge of SQL, as that is how both read and write access to databases is performed.

ADO is installed with a number of Microsoft products, such as SQL Server, IIS, and Internet Explorer. If your system doesn't have ADO, or has an out-of-date version, you can download MDAC (Microsoft Data Access Components) from http://www.microsoft.com/data. This will install a complete set of ADO, OLE DB, and ODBC components with coordinated and compatible versions.

After the ADO object is started, you need to connect to a database using the connect command. Once connected, you can retrieve information about tables or table columns using the gettables and getcolumns commands.

SQL commands can be executed for reading and writing data, or to manipulate database structures, assuming the session user has proper permissions. The execute command executes a SQL command, while the getexecute executes a SQL command and returns the number of rows affected. In either case, if the SQL command returns rows of data, the getrow and getrows commands can be used to retrieve the data. The getcols command returns a list of field names from the last executed SQL command.

Commands and parameters

Connect *connectstring*
Connects to a database in preparation for processing. Connect strings are passed to the database driver for parsing. They generally contain a data source name, a user ID, and a password. In some cases there may be no user or password required (an local Access database, for example). A database adminstrator should be able to provide the proper connection string for sdOffice sessions. Here are some examples:

Driver={SQL Server};server=bigsmile;uid=sa;pwd=pwd;database=pubs
DSN=Pubs;UID=sa;PWD=pwd
Data Source=Pubs;User ID=sa;Password=pwd

Close
Closes the open connection.

CloseRS

Closes an open record set derived from the last **execute** or **getexecute** command. Close a record set to regain data manipulation access to a table.

Execute *SQL command*
Executes the SQL command given. If the command returns rows, such as a SELECT statement, then subsequent **getrow** and **getrows** commands will return the data, and the **getcols** command will return a list of column names for the data.

GetCols
Returns a list of column names associated with the last **execute** or **getexecute** command. The column names are returned in the same order as the data in a **getrow** command.

GetColumns *tablename, wildcard, columnnames*
Returns the columns for a given table. The column records shown are defined by the wildcard. For example, **getcolumns customers,*sales** will show all columns ending with "sales" in the table "customers". The data returned included a header row of column names followed by any number of rows with column data. The columns shown, such as name, description, data type, and so on, can be specified by listing column names separated by commas. To see a list of valid columns, issue a command that will return no rows, such as **getcolumns customers,xxx**. A heading row will be followed by at most one row of column data.

The command **getcolumns products,*,column_name,data_type,numeric_precision** will return a list of columns in the "products" table, with each row containing the column name, data type, and precision.

GetExecute *SQL command*
This is identical to the **execute** command, except that the number of rows affected is returned. The count of affected rows is normally only returned from SQL commands that update data, such as INSERT commands.

GetRow
Returns the data from the next row returned from the last **execute** or **getexecute** command. The columns returned are determined by the content of the SQL command executed. Each column is separated by the current separator, which defaults to a comma. Any field data that contains the delimiter is quoted.

If the end of the rows is reached, then a single asterisk (*) is returned.

GetRows
Returns all remaining rows available from the last **execute** or **getexecute** command. The rows are prefixed by a header row containing column names.

GetTables *wildcard,columnnames*
Returns a list of tables whose names match the wildcard. The default wildcard is *, which matches all table names. Following the wildcard may be one or more column names separated by commas. If no column names are provided, then all columns are returned for the tables. The column names are used as headers in the first row returned, so an easy way to see a list of valid column names is with a command that returns no tables, such as **gettables xxx**, which will return a row of column names followed by at most one row (the xxx table, if it exists). The command **gettables *,table_name,description** will return a series of two-column rows containing the table name and description for each table in the database.

SetDelim *delimiter*
Sets the delimiter used when returning data with the various get commands. You can quote the delimiter if it contains spaces. You can use the character strings "\t" or "\n" to specify a tab or line-feed, respectively.

MAPI Automation

MAPI is Microsoft's messaging protocol. It is supported by many email client applications, such as Exchange, Outlook, and Outlook Express, and is included with all 32-bit Windows operating systems. sdOffice supports a MAPI object to provide generic email sending capabilities for users. Using a MAPI object, an application can create and send email, with attachments, from any workstation that has email configured. Outlook Automation also supports email, and additionally provides access to the user's email folders.

After MAPI is started, the application must start a session by signing on to an email profile using the signon command. Once a session is active, you can send email. Email profiles are maintained by the Mail and Fax program available from the Control Panel.

To send email, use the newmail command, optionally followed by an number of updatemail commands to add mail elements such as attachments and various types of recipients. When the mail is ready, it is sent with the send command.

Note that boolean parameters are true if present, false if not, in any command.

Commands and parameters

NewMail *parameters*
Creates a new email item, optionally setting elements based on the parameters. See the **updatemail** command for supported parameters and values. The email isn't sent until the **send** command is used.

Send or **SendMail** *parameters*
Sends the current email message, optionally using the following parameters:

- dialog or ask (boolean)

The dialog option will cause a send dialog window to be presented to the user on the sdOffice workstation before sending the email, if supported by the system's email system.

SignOn *parameters*
This command, which logs into an email profile, is required before any email can be sent or retrieved. Email profiles are defined with the Mail and Fax program, available from the Control Panel window.

- UserName or Profile=*name*
- Password=*password*
- Dialog or Ask (boolean)

A password may be optional, depending on the profile. Dialog will prompt the user at the sdOffice workstation for the username and password (or just a profile, depending on the configuration), then start the session.

SignOff
Ends the current session.

UpdateMail *parameters*
Updates the current email item with data found in the parameters. The fields and values for the parameters can be:

- To=*address*
- CC=*address*
- BCC=*address*
- Subject=*text*
- Body=*text*
- Attach=*pathname*

Addresses are resolved as encountered. They can be either Internet-style addresses or names from the user's address book. Any number of each type of address can be added with multiple to, cc, or bcc parameters.

The body text can contain tabs or line-feeds with "\t" and "\n" character sequences.

Attachment path names are relative to the sdOffice workstation. Multiple files can be attached with multiple attach parameters. An asterisk in the pathname is replaced with the sdOffice directory.

Microsoft Excel Automation

Excel uses a two-level document hierarchy. The first level is a workbook (or book), which is equivalent to a .xls file. An Excel session can have any number of workbooks open at one time (OpenBook). When you create a new book (NewBook), Excel names it Book*n*. You can then perform a SaveAs to give it a file name. Within each workbook are worksheets (or sheets). The worksheets contain the rows and columns of data. Worksheets are named, like workbooks, but the names are not related to the file name of the .xls file. When you create a new sheet (NewSheet), you may provide a name at that time as a parameter.

sdOffice works with an *active worksheet*. When you open or create a workbook, the first worksheet in that book is automatically activated. When you create a new sheet in the book, it is automatically activated. You can also manually activate a book (ActivateBook), and a sheet within the book (ActivateSheet), using their names. Sheets can also be cleared of their contents, or deleted entirely (ClearSheet, DelSheet). The name of the current book or sheet can also be obtained using GetBook or GetSheet.

You can retrieve the data from a worksheet with GetData. You can write data to the sheet with WriteCell or WriteRow. WriteCell provides full control over which cell gets updated, while WriteRow is an efficient way of writing any amount of data. WriteRow always writes from column 1, at a current row pointer. You can set the row pointer with SetRow.

You can format cells, columns, or rows with the Format command, and merge cells with the MergeCells command.

You can delete and insert columns and rows, using DelCol, DelRow, InsertCol, and InsertRow. You could use this capability to add a title row after sorting and subtotaling a list.

Print or manage the printer with Print, Printer, and PrintPreview. To change the page format, use PageSetup.

Once a sheet has been populated with data, you can sort the data on up to three columns. If you also have column headings in the first row, you can generate subtotals and grand totals based on breaks in a column.

Note that boolean parameters are true if present, false if not, in any command.

Command Usage and parameters

Activate or ActivateBook *workbook*

Activates an open workbook named as the parameter. The name is case-insensitive.

ActivateSheet *worksheet*
Activates the sheet named as the parameter. The name is case-insensitive.

AddChart *parameters*
Adds a new chart to the current worksheet. The new chart becomes the current chart. Use the **setchart** command to change the current chart. See the **editchart** command for a parameter description.

ClearSheet
Removes data and formatting from the current sheet.

CloseBook
Closes the active workbook without saving (use Save or SaveAs to save workbooks). After the command, you must use OpenBook, Newbook, or Activate to make a new workbook active.

DelCol *column*
Deletes the column number in specified as the parameter.

DelRow *rownum*
Deletes the row number specified in the parameter, and sets the current row to this value.

DelSheet
Deletes the active sheet from the active workbook. If the workbook has other sheets, the first sheet is activated.

EditChart *paramters*
Edits the current chart (see **addchart** or **setchart**) based on the parameter values specified. Unspecified parameters remain unchanged in the chart.

- x=*measure*
- y=*measure*
- w=*measure*
- h=*measure*
- Range=*cell range*
- Type=*chart type*
- Title=*chart title*
- CategoryTitle=*category axis title*
- ValueTitle=*value axis title*
- ExtraTitle=*extra title*
- ByColumn (boolean)
- ByRow (boolean)

- CatLabels=*cols or rows*
- SeriesLabels=*cols or rows*
- Legend=yes|true|no|false
- ApplyLabels=none|value|label|percent|labelpercent

Measures are used to define the size and location of the chart when displayed in the worksheet. The default location is 0,0 (upper left of worksheet), and the default width and height are 4 inches and 3 inches, respectively. Measures default to inches, but the units can be changed with the **units** command.

Cellranges are used to supply the data to chart. Charts use data in the worksheet on which they are added. The default range is the contigous data area starting with cell A1. To specify a different range, use an absolute range, such as "$B$1:$D$10", or a relative range from the current row. Excel will attempt to determine the descriptions and values from the range. This interpretation can be controlled by the CatLabels and SeriesLabels parameters, and the ByRow and ByColumn parameters.

The chart type can be one of the following names:
- area
- bar
- stackedbar
- 100bar
- column
- line
- stackedline
- pie
- radar
- xyscatter
- 3darea
- 3dbar
- 3dstackedbar
- 3d100bar
- 3dcolumn
- 3dline
- 3dpie
- 3dsurface
- doughnut

The various titles apply to the chart or axes. The extra title is used for some chart types.

Byrow and Bycolumn determine how Excel inteprets the worksheet range for data. Byrow is the default, where each row represents a new data series. Bycolumn interprets columns for the data series. When determining the series and category titles, Excel will analyze the worksheet range. You can specify the number of columns or rows to interpret using Catlabels and Serieslabels parameters.

ApplyLabels controls the use of labels on series data.

Format *parameters*
Use this command to format a cell, a column, a row, or the whole sheet. Formatting can include font information, alignment, width, height, and masking.

The following fields can be set with any number of *name*=*value* pairs in the parameter.

- autofit (boolean, no *value* needed)
- backcolor=*colorname*
- center (boolean)
- col=*column*
- color=*colorname*
- font=*name*
- fontbold or bold (boolean)
- fontitalic or italic (boolean)
- fontsize or size=*size*
- height=*measure*
- left (boolean)
- numberformat=*format*
- range=*range*
- right (boolean)
- row=*row*
- width=*measure*

If col and row are specified, then just the intersecting cell is affected. If col or row is specified, then the specified column or row is affected. If range is spacified (such as A1:F1), then all cells in the specified range are affected. If neither column nor row nor range is specified, then all cells are affected.

Measure values are given in inches by default, but the the unit if measure can be changed to points, millimeters, or centimeters with the Units command.

NumberFormat matches the number format values available in the Excel Format Cells dialog. To force text, use "@". This is useful for fields that appear numeric, such as zip codes or numeric ID codes, but which should be left justified. Date formats are also specified this way, though Excel recognizes most human-readable dates, such as "12/31/01", correctly. See Excel help for complete formatting instructions. Some example values:

- #,##0.00 (2 decimals with commas)
- m/d (short month/year)
- @ (text)
- 0.000 (3 decimals)
- General (general format)
- 00000 (zip code)

- mm/dd/yy (date)
- (* #,##0.00_);_(* (#,##0.00);_(* ""- ""??_);_(@_) (custom format)

Usage notes: Autofit should be performed after the data has been added to the cells. If text fields contain numeric data with leading 0s, like zip codes or ID codes, format the column as text (numberformat=@) before adding data. Otherwise, Excel assumes the data is numeric and removes the leading 0s.

Formats *colformats*
Sets a series of column numberformats, as found in the format command. Each column format is delimited by a tab or other column delimiter specified by the **setdelim** command. For example, to set columns 1 and 2 to text and columns 4 and 5 to a 2-decimal point number, with the delimiter set to a vertical bar (|), use this command: formats @|@||#,##0.00|#,##0.00. Note the third column is blank, and no numberformat is applied.

GetBook
Returns the name of the current workbook.

GetBooks
Returns all book names in the Excel session, each terminated with a CR-LF sequence, and ending with a single period (.CR-LF). If no books are opened, a asterisk is returned rather than one or more book names.

GetData
Returns data values from the current sheet. For a single cell, specify both *column* and *row* numbers. For a column, with each value delimited by CR-LF sequences, specify just *column*. For a row, with each value delimited by tab (CHR(9)) characters, specify just *row*. For all data, don't specify either *column* or *row*. Each row is returned delimited by CR-LF sequences. Within each row, each column is delimited by tabs.

Col=*column*
Row=*row*

GetRow
Returns the current row number where the next **writerow** will place data. Use **setrow** to modify the current row.

GetRows
Returns the number of rows in the contiguous non-empty region starting at cell A1. This can be used to determine where to append to a worksheet, as long as the worksheet has contiguous data, by using **getrows**, followed by **setrow** ***rows*+1**.

GetSheet

Returns the name of the current worksheet.

GetSheets
Returns all sheet names in the active workbook, each terminated with a CR-LF sequence, and ending with a single period (.CR-LF). If no book is open, or no sheets are in the active book, a asterisk is returned rather than one or more sheet names.

Hide
Hides the Excel window. To make it visible again, use Show. The window is hidden by default when the session starts.

InsertCol *colnum*
Inserts a new column at the column number parameter.

InsertRow *rownum*
Inserts a new row at the row number at the number parameter. The new row becomes the current row.

LeaveOpen
Normally, sdOffice.exe will close Excel when the session is ended. If you send this command, then Excel will be left open when the session ends.

MergeCells *parameters*
Merges multiple cells into one. This is useful to add title cells to reports. Only the value in the upper-left cell is retained and/or displayed when cells are merged.

Specify the upper left column and row as col and row, and the lower right column and row as col2 and row2.

- Col=*leftcolumnnum*
- Row=*toprow*
- Col2=*rightcolumnnum*
- Row2=*bottomrow*
- Range=*range*

Row2 defaults to row, and col2 defaults to col.

If Range is specified, the cells in the named range are merged, and any other parameters are ignored.

NewBook
Creates a new workbook. The name of the workbook is supplied later in a SaveAs command.

NewSheet *sheetname*
Adds a new sheet to the current workbook. If there is a text parameter, the sheet is so named.

OpenBook *workbook*
Opens a workbook (.xls file) named in as the parameter. An asterisk (*) in the file name will be substituted with the sdOffice path. For example, *salestable.xls would find the file in the sdOffice directory.

PageSetup *parameters*
Sets several page size options for the Excel environment. These options can affect the selection of printer characteristics automatically.

- Fitwidth (boolean)
- Gridlines (boolean)
- Orientation or orient=landscape|portrait
- Pagesize=*pagesize*

Fitwidth causes Excel to try to scale columns to fit the width of paper. If there are many columns, it may help to specify landscape orientation.

Gridlines causes Excel to add grid lines when printing.

Pagesize is one of several internal paper size names.

Print
Prints the current sheet.

Printer *parameters*
Sets the printer name and characteristics to values defined in the parameter text. Valid parameters are:

- Collate (boolean)
- Copies=*copies*
- From=*from page*
- Name=*printername*
- To=*to page*

Collate will turn on collation for multi-copy output. From and To determine a range of pages to print, referring to printed pages rather than worksheet pages. The *printername* must match a printer name in the list of system printers where sdOffice is running.

PrintPreview
Launches the Print Preview screen in Excel.

Run *macroname*
Runs the Public Sub-style VBA macro named as the parameter.

Save
Saves the active workbook.

SaveAs *workbook*
Saves the active workbook as the name supplied as the parameter. An asterisk (*) in the file name will be substituted with the sdOffice path. For example, *SalesTable would save the workbook in the sdOffice directory as SalesTable.xls.

ScreenUpdating *parameter*
Sets screen updating based on the parameter value. Off, No, or False will turn off screen updates until the session is closed, an error occurs, or another ScreenUpdating command is issued. Any other value turns screen updating on. Turning screen updating off can improve application performance.

SendKeys *keys*
Sends keystrokes to the application as if typed by the user from the keyboard. In order to send keys, the application window must be visible, so be sure and issue a Show command prior to this, or an error will be returned. In addition to standard text, there are many special keys and key combinations that can be entered by using special SendKeys characters..

SetChart *chartnumber*
Sets the current chart to the chart number specified. As charts are added, they are numbered starting with 1.

SetDelim *delimiter*
Sets the delimiter used by the **writerow** command to the text value *delimiter*. The default value is "\t", a text representation for the tab character. If desired, this can be set to some other character, such as "," or "|", to make writerow commands easier.

SetRow *rownum*
Sets the current row, used by WriteRow, to a new value. When a sheet is created or activated, the current row is set to 1.

Show
Make the Excel window visible. To hide the window, use the Hide command. If the application is left running with the LeaveOpen command, the window automatically becomes visible when the session closes.

Sort *parameters*

Used to sort the data in a sheet on values in up to three columns. You can specify up to three col= *column* values in the parameter text. If any column should be sorted in descending order, specify the col=*column* value, then the descending flag. If Header is specified, then the first row in the sheet is assumed to be column headings and is not sorted.

- Col=*column*
- Descending or Dsnd (boolean)
- Header (boolean)

SubTotal *parameters*
This function can be used to add Excel-generated sub-totals to a sheet. The sheet must be in contiguous columns, with a heading row at the top, or an Excel error occurs.

To add subtotals, you choose one column to be the "group by" column, a summary function, and any number of columns on which to apply the function. Whenever the "group by" column changes, a sub-total line is inserted with the appropriate function applied to the sub-totaled columns. Grand totals are also applied at the end of the sheet.

- Above (boolean)
- Below (boolean)
- Col=*column*
- Function=*functionname*
- Group=*column*
- PageBreak (boolean)
- Replace (boolean)

Function names can be:
- Avg
- Count
- Countnums
- Max
- Min
- Product
- StdDev or Std
- Sum
- Variance or Var

Multiple Col values can be specified, but only one Group and Function are allowed. The subtotal function is applied to all columns specified. For example, "col=2, col=4, col=5, function=sum" will sum columns 2, 4, and 5.

Above will generate subtotals above the group of rows to which they apply. Below generates the subtotals below, the default.

If you specify PageBreak, then a print of the sheet will generate page breaks at group break points.

Replace will cause Excel to replace any existing subtotals in the sheet with the new ones. The default is to add new subtotals, allowing for a series of sub-totals to be generated for different columns or functions.

Units *unitname*
Sets the unit of measure for subsequent measure values. The default unit of measure is inches. It may be set to any of these values:

- points or pts or p
- millimeters or mm or m
- centimeters or cm or c

All other values are interpreted as inches.

WriteCell *parameters*
This will write the value (number, date, or text) or formula (=*expression*) specified in the cell specified. The following parameters are required:

- Col=*column*
- Range=*range*
- Row=*row*
- Value=*value*

To set a specific cell by column and row number, specify both Col and Row values. Optionally, specify a range, such as F2:F30, to assign all cells in the range to the same value or formula. A special character in the range of "*" will be substituted with the current row. F2:F* will represent the range F2:F30, if the current row is 30.

WriteRow *parameters*
Writes one or a series of rows, starting at the current row, with parameter data supplied. The data columns are delimited by tab characters, supplied as either tab characters - CHR(9) - or "\t" strings, or by the character specified in a previous **setdelim** command. Each row is delimited by a "\n" sequence.

While it is possible to write formulas to cells in this manner, each row would have to be adjusted to ensure correct relative addressing. To write formulas to a range of cells, it is easier to use the WriteCell function, as Excel handles relative cell addressing automatically.

Microsoft Outlook Automation

This program works with Outlook appointments, contacts, tasks, and email folders. After starting the automation session, you can select a folder to work with using the SetFolder command. When adding the various types of records, an appropriate default folder will be automatically selected if necessary. This current folder is used for the SetGet and GetNext commands. To view valid folders, use the GetFolders command.

You work with current appointments, contacts, tasks, or emails. The current record is specified with an New*item* command, such as as NewTask or NewAppointment, or with a GetNext command, which follows a SetGet command, which defines search criteria and return data for the current folder. To modify the data in any current record, use the Edit*item* commands, such as EditAppointment. To delete a record, use the Del*item* commands, such as DelContact.

Note that boolean parameters are true if present, false if not, in any command.

Command Usage and parameters

DelAppointment or **DelAppt**
Deletes the current appointment record. The current appointment record is normally selected via a setget/getnext sequence. After the delete, there is no current appointment record.

DelContact
Deletes the current contact record.

DelTask
Deletes the current task record.

GetFolders
Returns a list of folders available in the user's Outlook configuration. Each folder consists of a comma-delimited path, such as "Personal Folders,Calendar". Multiple folders are delimited by a CR-LF sequence. sdOffice will scan up to three levels deep in the user's folder hierarchy. Each folder name is suffixed by the type of records, Appointment, Contact, Email, or Task, in parenthesis.

GetAll
Returns all remaining records after a SetGet function. Each record has the same format as a GetNext response, and multiple records are delimited by an extra blank line.

GetNext
Makes the next available record after a SetGet function the current record, and returns a list of fields. If no more records match the criteria from the SetGet command, then a "*" is returned.

The data format returned for records is based on the field names specified in the last SetGet command. Each field is returned in the format *name=value*, with a CR-LF sequence delimiting each filed.

LeaveClose
Normally (and unlike the Word and Excel interfaces), sdOffice will leave the Outlook task running when the session closes. Issing this command will cause sdOffice to close the Outlook task when it closes.

NewAppointment or **NewAppt** *parameters*
Adds a new appointment, sets any field values defined as parameters, saves the appointment, and leaves it as the current appointment record. See the **UpdateAppointment** command for a list of valid fields. Subsequent Update commands can be used to update the same record.

NewContact *parameters*
Adds a new contact, sets any field values defined in the parameters, saves the contact, and leaves it as the current contact record. See the **UpdateContact** command for a list of valid fields. Subsequent UpdateContact commands can be used to update the same record.

Newmail or **Email** or **NewEmail** *parameters*
Creates a new email message, optionally setting certain message elements from parameter arguments. See the **UpdateMail** command for a list of valid parameters. Once the message is created, additional elements of the message can be updated with subsequent **UpdateMail** commands, until the **SendMail** command is used.

SendMail
Sends the current email, previously defined with a **NewMail** command, and optionally edited with the **UpdateMail** command. Once the mail is sent, there is no current email.

Usage note: In Outlook, immediate delivery must be enabled for email to be sent automatically. Within Outlook, choose the Tools menu, Options window. On the Options window, select the Email or Mail Delivery tab and enable the immediate or automatic delivery of messages. The terminology varies somewhat between different versions of Outlook.

SetFolder *foldername*

Sets the current folder to the parameter value . This must be a valid appointment, contact, email, or task folder, with the hierarchy levels delimited by commas; "Public Folders,Sales,Meetings", for example. You can also use one of these standard names to get the user's default folder of that type:

- contacts
- appointments
- email
- tasks

SetGet *parameters*
Sets criteria and fields for subsequent GetNext and GetAll commands. The parameters consist of a search expression, which always starts with a field name in brackets, such as [subject], followed by a list of field names to return in GetNext and GetAll commands.

Search Expression
The search expression format is defined by Microsoft as one or more boolean functions separated by And or Or. Field names from the appropriate database are placed inside square brackets, and are compared with literal values enclosed in quotes or plain numbers. Valid operators are similar to Basic operators: >, <, =, >=, and <=. The various Update commands list common field names that are available for the different record types. For example, if the current folder is an appointments folder, then the search expression could check the [Subject] field, the [Start] field, and others found in an appointment record.

Examples:

[Start]>="12/20/2000 9:00am"
[Start] > "12/20/2000" And [End]<="12/20/2000 6:00pm"

GetNext Fields
In addition to the search criteria, which is indicated by an opening square bracket, you should list one or more field names to return with the GetNext command. For a list of valid field names, see the lists below. Note that for search expressions, only the first name is valid, in cases where multiple field name options are listed.

Delimit each field, and the search expression, with commas.

A complete SetGet command will might look like this:

SetGet [Start]>="12/20/2000 6:00 AM",start,duration,subject

Valid field names for appointments:
- start
- end

- duration
- subject
- body
- location
- alldayevent
- reminder or reminderminutes
- entryid

Valid field names for contacts:

- fullname
- lastname
- firstname
- companyname or name
- businessaddressstreet or street
- businessaddressstate or state
- businessaddresscity or city
- businessaddresspobox or pobox
- businessaddresspostalcode or zip or postalcode
- businessaddresscountry or country
- businessphone or phone
- businessphone2 or phone2
- businessfax or fax
- businesshomepage or homepage
- email
- email2
- email3
- otherphone
- otherfax
- account
- customerid
- user1
- user2
- user3
- user4
- entryid

Valid field names for tasks:

- startdate
- duedate
- remindertime
- subject
- body
- reminder or reminderminutes
- entryid
- complete
- datecompleted
- status
- actualwork
- totalwork
- delegator
- percentcomplete

Valid field names for email:

- to

- cc
- bcc
- subject
- replyto
- attach
- body
- sendername
- senttime
- receivedtime
- entryid

UpdateAppointment or **UpdateAppt** *parameters*
Updates the current appointment record with parameter fields and values. The current record is specified by either a NewAppointment command or a SetGet/GetNext sequence.

The primary and alternate field names available are:

- alldayevent (boolean)
- body=*text*
- duration=*minutes*
- end=*datetime* (such as "12/20/2001 10:15am")
- location=*text*
- reminder (boolean)
- reminderminutesbeforestart or reminderminutes=*minutes*
- start=*datetime* (text date/time)
- subject=*text*

By default, reminders on new appointments are turned off. You can set Reminder or ReminderMinutesBeforeStart to turn on a reminder.

UpdateContact *parameters*
Updates the current contact record with parameter fields and values. The current contact record is specified by either a NewContact command or a SetGet/GetNext sequence.

The primary and alternate field names, all of which can be set to any text, are:
- account
- businessaddress
- businessaddresscity or city
- businessaddresscountry or country
- businessaddresspobox or pobox
- businessaddresspostalcode, zip, or postalcode
- businessaddressstate or state
- businessaddressstreet or street
- businessfax or fax
- businesshomepage or homepage

- businessphone or phone
- businessphone2 or phone2
- companyname or name
- customerid
- email
- email2
- email3
- firstname
- fullname
- lastname
- otherfax
- otherphone
- user1
- user2
- user3
- user4

UpdateMail *parameters*
Updates the current email message. You can issue any number of UpdateMail commands to prepare an email, then issue the **SendMail** command to send the email.

The parameter names recognized are:

- Attach=*pathname*
- Bcc=*address*
- Body=*text* (use \n for line breaks)
- Cc=*address*
- Replyto=*address*
- Subject=*text*
- To=*address*

The attach command can contain the name of a file to attach. Use multiple attach parameters to attach multiple files. Any asterisk (*) character in the pathname is replaced with the path to the sdOffice directory. Pathnames are relative to the sdOffice workstation.

The To, CC, ReplyTo, and Bcc fields can specify email addresses or address book aliases. Each is resolved as encountered. Multiple addresses can be entered with multiple parameters. For example, to add two CC address, use cc=*first*, cc=*second*.

UpdateTask *parameters*

Updates the current task record with parameter fields and values. The current task record is specified by either a NewTask command or a SetGet/GetNext sequence.

startdate=*datetime*
duedate=*datetime*
remindertime=*datetime*
subject
body
reminder=yes|true

Date/time values can be entered in any recognizable format, such as "12/31/2001 6:00PM" or "December 31, 2001 6:00 PM".

Microsoft Word Automation

The basic unit in Word is a document, which is equivalent to .doc file. You can open any number of documents in a Word session. sdOffice works with one document at a time, called the *active document*. To create a new document, use the NewDoc command. Give it a name with the SaveAs function. New documents can be based on an existing Word document template. To open an existing document, use the OpenDoc command. Both NewDoc and OpenDoc automatically set the active document. To get the name of the active document, use GetDoc; to activate any open document, use Activate.

In an active document, you can add paragraphs (Write), page breaks (NewPage), tables (Table and TableRow), bullet lists (BulletList), numbered lists (NumberList), and images (Image). The format of added text can be modified with Font and Paragraph commands. Table cell, column, and row formatting can be modified with TableDef.

If a document contains merge fields, you can place values in those fields with MergeField. To just replace text values with new values, use Replace.

You can copy the active document to the clipboard with CopyDoc, then paste either the rich text or plain text to another document with PasteDoc and PasteText. You can also clear the contents of the current document with ClearDoc.

To print the document, use Print, Printer, and PrintPreview. To modify the page setup, use PageSetup.

Note that boolean parameters are true if present, false if not, in any command.

Command Usage and parameters

Activate *documentname*
Activates the document named in the parameter text. This is normally the document file name, without leading path information. For a new document, it is usually "Document*n*". You can use GetDoc to retrieve the name of the active document.

BulletList *list*
Converts the contents of the parameter into a bullet-style list. Each paragraph, delimited by a "\n" sequence, becomes a bullet list item.

ClearDoc
Clears all the text from the current document.

CloseDoc
Closes the active document without saving it. Use Save or SaveAs to save the document. After this command, no document is active. Use OpenDoc, NewDoc, or Activate to make another document the active document.

CopyDoc
Copies the current document to the clipboard. Word copies both the plain text and the rich text forms. It does not copy headers or footers, just the story text.

Font *parameters*
Sets the font for any new text added to the document. The font defaults to the "Normal" style defined in the user's Word configuration. Each element of the font is defined with a *name=value* pair, with pairs delimited by commas.

- bold (boolean)
- color=*colorname*
- italic (boolean)
- name=*fontname*
- normal (boolean)
- size=*points*

Setting "normal" will revert all attributes to the Normal style defined in the user's Word configuration.

GetDoc
Returns the current document name. This name can be used by the Activate command.

GetDocs
Returns all document names in the Word session, each terminated with a CR-LF sequence, and ending with a single period (.CR-LF). If no documents are opened, a asterisk is returned rather than one or more names.

GetFields
Returns all the mergefield names in the current document, delimited by the current delimiter specified by the **setdelim** command. The default delimiter is a tab character (CHR(9)).

GetParagraph
Returns the current paragraph number. When a document is opened or activated, the current paragraph number is set to the document paragraph count. The paragraph number can be set with the SetParagraph command.

GetParagraphs
Returns the number of paragraphs in the document.

GetText
Returns the text of the current document. Paragraphs will be delimited by CR-LF sequences.

Hide
Makes the Word window invisible. To show the window again, use Show. The Word window is hidden by default when then session is started.

Image *parameters*
Adds an image to the current document. The image can be placed "in-line" as a new paragraph, or can be placed anywhere on the page that the new paragraph resides on.

- file=*imagefilename*
- h=*measure*
- inline (boolean)
- stretch (boolean)
- w=*measure*
- x=*measure*
- y=*measure*

The *imagefilename* value should be a full path to the image to be loaded. An asterisk (*) in the file name will be substituted with the sdOffice path. For example, *logo.jpg would find the file logo.jpg in the sdOffice directory.

x,y,w, and h are size and position values. The images is proportionally adjusted to fit, unless the stretch flag is present, in which case the image is stretched to fit the w and h dimensions.

If the inline flag is present, the image will remain between the preceding and next paragraphs when the image command is issued, with the x and y values being offsets from that paragraph position. Otherwise, the x and y values are absolute page positions.

Measure values are given in inches by default, but the the unit if measure can be changed to points, millimeters, or centimeters with the Units command.

LeaveOpen
Normally, sdOffice will close Word when the session is closed. If you send this command, Word will be left open.

MergeField *parameters*
Scans the document for merge fields named in the parameters, setting their values to each name's value. Set the field names and their associated values as

name=value pairs. The value can contain tab or CR-LF characters as "\t" and "\n", respectively.

sdOffice specifically looks for MERGEFIELD type codes, which can be inserted into a document with the "Insert, Field" dialog box, selecting "Mail Merge" type fields, and then selecting the "Merge Field" subtype.

NewDoc *documentname*
Adds a new document to the session, and makes it the current document. To retrieve the name, use GetDoc. If a name is provided as a parameter, it is used as a document template for the new document. The document template file must exist either as a full path or in the user's Templates directory. An asterisk (*) in the file name will be substituted with the sdOffice path. For example, *letterhead.dot would find the file letterhead in the sdOffice directory.

NewPage
Adds a page break to the current document. The current paragraph is set to begin writing at the top of the new page.

NumberList *list*
Converts the contents of the parameter text into a numbered list. Each paragraph, delimited by a "\n" sequence, becomes a list item.

OpenDoc *documentname,options*
Opens the document file named as the parameter, and sets that as the current document. An asterisk (*) in the file name will be substituted with the sdOffice path. For example, "*Collection Letter.doc" would find the file in the sdOffice directory.

An option "readonly" is available, allowing multiple users to open the same document without errors, as long as all users use the readonly mode.

PageSetup *parameters*
Sets several page size options for the Word environment. These options can affect the selection of printer characteristics automatically.

- BottomMargin=*measure*
- Height=*measure*
- LeftMargin=*measure*
- Orientation or Orient=landscape|portrait
- PageSize=*pagesize*
- RightMargin=*measure*
- TopMargin=*measure*
- Width=*measure*

pagesize can be one of several internal paper size names.

Measure values are given in inches by default, but the the unit if measure can be changed to points, millimeters, or centimeters with the Units command.

Paragraph *parameters*
Sets paragraph characteristics for paragraphs added after this command. Use this to set alignment, indentation, borders, shading, and other paragraph settings.

- Left (boolean)
- Right (boolean)
- Center (boolean)
- Justify (boolean)
- Indent or LeftIndent=*measure*
- RightIndent=*measure*
- Keeptogether (boolean)
- SpaceBefore=*measure*
- SpaceAfter=*measure*
- Shade=2.5|5|10|15|20|25|30|40|50|75|100
- Border (boolean)
- Normal (boolean)

Normal, if it appears as a parameter, will force all options to their normal state.

Measure values are given in inches by default, but the the unit if measure can be changed to points, millimeters, or centimeters with the Units command.

PasteDoc
Paste rich text from the clipboard to the end of the current document. Rich text can be placed on the clipboard by the CopyDoc command.

PasteText
Paste unformatted text from the clipboard to the end of the current document.

Print
Print the current document.

Printer *parameters*
Sets the printer name and characteristics to values defined in the parameter text. Valid parameters are:

- Collate (boolean)
- Copies=*copies*
- From=*from page*
- Name=*printername*
- Pages=*page range(s)*

- To=*to page*

Collate will turn on collation for multi-copy output. From and To determine a range of pages to print. Alternatively, specify the Pages parameter, which accepts a list of page numbers and/or page ranges, such as "1, 5-9" for pages 1, and 5 through 9. Remember to quote the range if it contains commas. The *printername* must match a printer name in the list of system printers where sdOffice is running.

PrintPreview
Executes a Print Preview of the current document in Word.

Replace *parameters*
Scan the current document for occurrences of names in the parameter text, replacing them with the associated values. Values can contain tabs and CR-LF values as "\t" or "\n", respectively. For example, **replace [Name]="Acme Incorporated"** will replace the text string "[Name]" with Acme Incorporated.

Run *macroname*
Runs the Public Sub-style VBA macro named as the parameter.

Save
Saves the current document. Note that the document must already be named with a previous **OpenDoc** or **SaveAs** command.

SaveAs *documentname*
Saves the current document as the name specified in the parameter text. An asterisk (*) in the file name will be substituted with the sdOffice path. For example, *Collections would save the file as Collections.doc in the sdOffice directory.

ScreenUpdating *parameter*
Sets screen updating based on the parameter value. Off, No, or False will turn off screen updates until the session is closed, an error occurs, or another ScreenUpdating command is issued. Any other value will turn screen updating on. Turning screen updating off can improve application performance.

SendKeys *keys*
Sends keystrokes to the application as if typed by the user from the keyboard. In order to send keys, the application window must be visible, so be sure and issue a Show command prior to this, or an error will be returned. In addition to standard text, there are many special keys and key combinations that can be entered by using special SendKeys characters..

SetDelim *delimiter*

Sets the delimiter used by the **tablerow** command to the text value *delimiter*. The default value is "\t", a text representation for the tab character. If desired, this can be set to some other character, such as "," or "|", to make tablerow commands easier.

SetParagraph *number*
Sets the current paragraph to the parameter value. To force an append on the next Write, BulletList, NumberList, or Table command, set the paragraph to an artifically high number.

Show
Make the Word window visible. To hide the window, use the Hide command. If the application is left running with the LeaveOpen command, the window automatically becomes visible when the session closes.

Table *parameters*
Adds a table to the current document, using the characteristics specified as parameters. Once the table is defined, you can add rows to it with the TableRow command, and define individual cell, column, or row characteristics with the TableDef command.

- Autofit (boolean)
- Borders or Border=grid|box
- Center (boolean)
- Cols=*colwidths*
- HeadRows=*rows*
- Left (boolean)
- Right (boolean)

The left, right, and center options align the table within the page margins. If autofit is used, then the table columns will be sized automatically (if possible) to fit the cell data. The table can have an outside box, or an outside box and inner grid lines, based on the setting of Borders. HeadRows instructs Word to re-display the top *rows* at a page break.

You can pre-define the number of columns and their widths with the Cols option. Set *colwidths* to a space-delimited list of widths in the current unit of measure (default=inches). For example, cols="1.5 3.25 1.25" would create a three-column table, with the widths 1.5, 3.25, and 1.25 inches.

TableDef *parameters*
Sets the cell characteristics of a given cell, column, or row in the last table defined in the document. New tables are created with the Table command.

- BackColor=*colorname*
- Center (boolean)

- Col=*column*|last
- Color=*colorname*
- Font=*fontname*
- FontBold or Bold (boolean)
- FontItalic or Italic (boolean)
- FontSize or Size=*points*
- Height=*measure*
- Left (boolean)
- Right (boolean)
- Row=*row*|last
- Width=*measure*

If you specify both Col and Row, then the specified cell is affected. If you specify just Col or Row, then the specified column or row is affected. Otherwise, all cells are affected.

You can specify "last" for either Col or Row, and the bottom row or right-most column, at the time of the command, is used.

Left, right, and center options will align the cells specified.

Rows and columns are added as necessary. If a column is added that would exceed the width of the page, an error will occur.

Measure values are given in inches by default, but the the unit if measure can be changed to points, millimeters, or centimeters with the Units command.

TableRow *text*
Adds one or more rows from the parameter text to the last table in the document. Each row is delimited by the text sequence "\n" or a linefeed character (CHR(10)). Each cell within the row is delimited by the current delimiter set by the **setdelim** command. The delimiter defaults to a tab (CHR(9) or "\t").

Rows and columns are added as necessary. Take care not to add columns that would exceed the page margins, or an error will result. It may be necessary to specify column widths in the initial Table command or in previous TableDef commands.

Units *unitname*
Sets the unit of measure for subsequent measure values. The default unit of measure is inches. It may be set to any of these values:

- points or pts or p
- millimeters or mm or m
- centimeters or cm or c

All other values are interpreted as inches.

Write or **Type** *text*
Adds one or more paragraphs from the parameter text to the end of the current paragraph. The current paragraph is incremented by the number of paragraphs added. To append to an existing pararaph, use **setparagraph** before the write command.. Normally, paragraphs will be delimited by two "\n" sequences, and Word will word-wrap the text based on the current page, paragraph, and font settings.

In addition to the paragraph breaks, you can insert tabs with CHR(9) characters or "\t" sequences.

About the Samples

sdOffice comes with a number of samples which can be referenced for "how to" information. Each sample is provided in the sdOffice directory in a plain text file. All files, and associated documents that are used by the samples, start with the characters "s_". The *.txt files are each command files that can be used with the sdRun programs.

To run these samples, choose one of these methods, depending on your environment:

Unix

- Copy the samples (s_*.*) to a directory on your Unix system.
- Start the sdOffice server on your Windows workstation.
- Using a terminal emulator, login to the Unix system using a TCP/IP protocol such as telnet. sdrun.pl can then determine your workstation's address.
- Use the perl script sdrun.pl for each sample: **perl sdrun.pl *SampleFile***

Optionally, you can send the commands to another workstation running the sdOffice server by appending the server IP address or hostname and port to the perl command.

Windows

- Open a MS-DOS command window.
- cd to "c:\program files\sdsi\sdoffice" (or other directory if you installed sdOffice elsewhere).
- Run the samples using sdrun.exe: **sdrun *SampleFile***

If you want to direct the commands to another workstation running the sdOffice server, append the server IP address or hostname and port to the sdrun command.

PRO/5, Visual PRO/5, or ProvideX

On a local Windows workstation, the sdOffice server need not be running, as these these programs use the DDE interface. On Unix, the sdOffice server must be running, and you need to be logged into the Unix system as a terminal user over TCP/IP.

From the Basic console prompt (usually ">"), enter **call "sdrun.bb","*SampleFile*","","",resp$,errmsg$**

For ProvideX, use "sdrun.pv" in place of "sdrun.bb". If you want to direct the commands to another workstation, change the two null ("") arguments to "*ServerIP*" and "*ServerPort*", respectively.

Sample: ADO Database Manipulation

File: s_ado.txt

```
# This sample shows some of what you can do with the ADO object,
# used for database manipulation through ODBC, OLE DB, and ADO
# providers.  This example assumes you have a data source called
# NorthWind, which is installed as a sample with Microsoft Access.

# Start ADO
ado

# Connect to the database.  Most databases will require several
# parameters in the connect string, such as DSN, UID, and PWD.
# This local Access database doesn't require everything.
connect dsn=NorthWind

# Return a list of tables with the word Product in the name.  The
# list returned contains two columns.
gettables *Product*,table_name,description

# Issue a query, then return all the rows.
execute select * from products
getrows

# Create a new table
execute create table TestTable (id char(5), name char(30))
execute insert into TestTable (id,name) values ('10','Test Record 10')
execute insert into TestTable (id,name) values ('20','Record 20')

# show columns of table
getcolumns TestTable,*,column_name,data_type,character_maximum_length

# Now query that table and return records one at a time.  When the end
# of records is encountered (the third getrow), an * is returned.
execute select name,id from TestTable order by name
getcols
getrow
getrow
getrow
closers

# Remove the test table
execute drop table TestTable
```

Sample: Excel Calculation Engine

File: s_excel2.txt

```
# This example uses an existing worksheet macro to return a calculated value.

# Start Excel.  It will remain hidden.
excel

# Open workbook s_excel2.xls in sdOffice directory.
# This workbook contains a macro that calculates the Standard Deviation
# of values in the first column (according to MS Office specifications,
# no more than 30 values can be used).  The result is placed in cell B1.
# To view the macro, open the workbook in Excel and use Tools, Macro.
openbook *s_excel2.xls

# Write some values to column 1.
writerow 12\n15\n99\n85

# Run the macro.
run CalcStdDev

# Return the result
getdata range=b1
```

Sample: Excel Formatting

File: s_excel1.txt

```
# This sample demonstrates many of the formatting capabilites you can
# use when writing Excel worksheets.

# Start Excel and open a new sheet.
excel
newbook
newsheet Formatting

# Write some data.  Each \t represents a tab to a new cell.  You could
# add \n to break to the next row.  Sequential commands automatically
# do a row break.
writerow Text Column\tZip Code\tNumeric Text\tAmount 1\tAmount 2
writerow Text 1\t95682\t134.50\t1111.11\t-2222.22
writerow Text 2\t00222\t955.25\t10.12\t0

# Change format of text column.  Units for width default to inches.
format col=1,width=2.0,font=Courier,italic,backcolor=blue,color=white

# Force zip code to be 00000, left justified.
format col=2,numberformat=00000,left

# Force column 3 to text style.
format col=3,numberformat=@

# Format the last two columns as numbers, using the Range option.
format range=d:e,numberformat="#,##0.00;(#,##0.00)"

# Fix heading row, which overrides previous formats that affected row 1.
format row=1,backcolor=gray25,bold,size=12,color=black

# Autofit that whole sheet.
format autofit

# Leave Excel running when we exit.  Upon exit, Excel will become
# visible automatically.  We could explicitly force this with a show
# command.
leaveopen
```

Sample: Excel Report

File: s_excel3.txt

```
# This sample generates a report with 31 data lines.  It then formats,
# sorts, and sub-totals the report.  Finally, it adds a report title.

# Start excel and open up a new worksheet.
excel
newbook
newsheet

# Watch what's going on, though in practice, performance is better if
# the window remains hidden.
show

# Write some data.  At least here, we turn off screen updating for the writes.
# But first, write a heading row.
writerow Slsp\tSlsp Name\tCustID\tCustomer Name\tYTD Sales\tYTD Cost
screenupdating off
writerow 100\tSALLY SMITH\t00005\tADVANTAGE BUSINESS FORMS\t10514.85\t1752.48
writerow 110\tGEORGE WINSTON\t00013\tALLIED SERVICES, INC.\t8705.61\t1088.2
writerow 110\tGEORGE WINSTON\t00018\tEAGLE FORMS\t18712.21\t3742.44
writerow 110\tGEORGE WINSTON\t00026\tWESTERN COMPUTER
SERVICES\t12514.85\t2502.97
writerow 101\tJERRY JONES\t00030\tMARCH, INC.\t8906.27\t1781.25
writerow 100\tSALLY SMITH\t00037\tPROFESSIONAL HELP SVC.\t6504.95\t1300.99
writerow 100\tSALLY SMITH\t00042\tALL-PRO FORMS\t17106.93\t2138.37
writerow 100\tSALLY SMITH\t00046\tROCKY MOUNTAIN MANAGEMENT\t17003.3\t2429.04
writerow 110\tGEORGE WINSTON\t00055\tWASHINGTON ST.
COMPUTERS\t9106.93\t1517.82
writerow 101\tJERRY JONES\t00064\tSOUTHWEST INVESTMENTS\t9702.31\t1212.79
writerow 101\tJERRY JONES\t00073\tGREEN & GREEN, INC.\t4909.57\t981.91
writerow 110\tGEORGE WINSTON\t00080\tGREAT LAKES MANAGEMENT
LTD.\t14110.23\t2822.05
writerow 101\tJERRY JONES\t00084\tEMPIRE COMPUTERS &
SOFTWARE\t13407.92\t2234.65
writerow 101\tJERRY JONES\t00088\tMBA\t12705.61\t2117.6
writerow 110\tGEORGE WINSTON\t00094\tBUSINESS RESOURCES, INC.\t19909.57\t3981.91
writerow 110\tGEORGE WINSTON\t00112\tALLANTE SYSTEMS INC.\t6013.2\t859.03
writerow 101\tJERRY JONES\t00116\tRIORDAN COMPANY\t3705.61\t529.37
writerow 101\tJERRY JONES\t00119\tBROWNIE'S COMPUTERS\t16514.85\t2359.26
writerow 110\tGEORGE WINSTON\t00128\tABLE PLUS, INC.\t5611.88\t935.31
writerow 110\tGEORGE WINSTON\t00131\tELKHORN PLAZA COMPUTERS\t13006.6\t2601.32
writerow 100\tSALLY SMITH\t00135\tSPECIALTY CONSULTING, INC.\t2000\t250
writerow 101\tJERRY JONES\t00152\tJL MARKET SERVICES\t3401.32\t680.26
writerow 101\tJERRY JONES\t00161\tABC BUSINESS FORMS\t20712.21\t4142.44
writerow 100\tSALLY SMITH\t00170\tZEBRA FORMS, INC.\t19508.25\t2438.53
writerow 110\tGEORGE WINSTON\t00175\tCLINKERDALES SUPPLIES\t11000\t1833.33
writerow 110\tGEORGE WINSTON\t00181\tJENSEN COMMERCIAL LTD.\t8100.33\t1012.54
writerow 101\tJERRY JONES\t00184\tBRIDON SERVICES\t22712.21\t3244.6
writerow 101\tJERRY JONES\t00191\tABLE CONSULTING\t18203.96\t3033.99
writerow 101\tJERRY JONES\t00205\tWALKER & WADE\t5003.3\t1000.66
writerow 100\tSALLY SMITH\t00210\tRUSTY'S BUSINESS FORMS\t5511.55\t918.59
```

```
screenupdating on

# Add a gross profit calculation to column G using a formula.  The range
# g2:g* represents rows 2 through the current row.  The current row is
# actually the next one to be written, so after filling the cells with
# the formula, we reset the last one, which is below the written range,
#  to "".
writecell range=g1,value="YTD Profit"
writecell range=g2:g*,value="=e2-f2"
writecell range=g*,value=""

# Now sort the data first on column 1 (Slsp ID), then descending column
# 5 (YTD Sales).
sort col=1,col=5,descending,header

# Add subtotals.  To do this, that heading row written earlier is required.
subtotal group=1,col=5,col=6,col=7,function=sum

# Do some formatting.
format col=1,numberformat=@
format col=3,numberformat="00000",left
format range=e:g,numberformat="#,##0.00"

# Dress up the heading
format row=1,color=blue,backcolor=gray25,bold

# Make everything fit.  By not selecting any range, the whole sheet gets
# affected.

format autofit

# Insert a title
insertrow 1
writecell col=1,row=1,value="Customers by Salesperson"
format col=1,row=1,font=New Times Roman,size=14,bold,center
mergecells range=a1:g1

# Don't close Excel when we're done.
leaveopen
```

Sample: Excel Charting

File: s_excel4.txt

```
# This sample demonstrates a worksheet with two charts, by first
# creating some worksheet data, then adding a column chart and a
# pie chart.

excel

# Open a new workbook and show the process
newbook
show

# Set the delimiter from the default tab (or \t) to a comma,
# simplifying the next couple of statements, which write data
# to the worksheet.
setdelim ,
writerow Name,Sales,Cost
writerow Allen,200,100
writerow Bill,250,123
writerow Sue,260,175

# Add a 3D column chart at the position 3,0.25 (inches).  Edit some
# additional chart values with additional editchart commands.
addchart x=3,y=.25,type=3dcolumn
editchart title="Sales Figures"
editchart valuetitle="Amounts in Dollars"

# Add a pie chart, using data from the first two columns only, and
# interpreting the value groups by column rather than the default,
# by row.  Make the chart smaller than the default 4x3 inches.
addchart y=1,x=.1,w=2.5,h=2.5,type=pie,range=$A$1:$B$4,bycolumn
editchart title="Sales Figures"
editchart applylabels=percent
leaveopen
```

Sample: MAPI email submission

File: s_mapi1.txt

```
# This sample uses MAPI to create an email with attachments and
# send it to a dummy email address.  Modify the address to send
# to yourself to see the result.

mapi

# MAPI sessions must be logged on.  This command will prompt the
# user for profile information and start the session.

signon ask

# Creates a new email, setting the To: address.  You can set
# parameters here or in subsequent updatemail commands.  Note the
# use of quotes around the subject to hide the embedded comma.
# To attach multiple files, just use multiple attach commands.
newmail to=allenm@synergetic-data.com
updatemail subject="A test of MAPI email automation, from sdOffice."
updatemail body="Line 1\nLine 2\nLine3\n\nFrom,\nMe\n"
updatemail attach=*sdrun.pl,attach=*s_mapi1.txt

# When the mail is ready, send it.
send
```

Sample: Outlook Add Appointment

File: s_appt1.txt

```
# This sample will write a test appointment to your Outlook Calendar
# database.  Note that each time you run this, an additional record
# will be added!

outlook

# This establishes the current appointment record as a new empty record,
# and sets the start time.  The record isn't actually written to Outlook
# until an update command is issued.

newappointment start="1/1/2001 8:00 AM"

# Update fields into the new record.  Each update will write the
# associated data to the same current record.  You can set field values
# either in the newappointment command or the update command.

updateappt duration=60,subject="Test entry from sdOffice"
```

Sample: Outlook Add Contact

File: s_cont1.txt

```
# This sample will write a test contact to your Outlook Contacts
# database.  The customer ID for the data will be "test".  Note that
# each time you run this, an additional record will be added!

outlook

# This establishes the current contact record as a new empty record,
# and sets the CustomerID field to "test".  The record isn't actually
# written to Outlook until an update command is issued.

newcontact customerid="test"

# Update fields into the new record.  Each update will write the
# associated data to the same current record.  You can set field
# values either in the newcontact command or the update command.

updatecontact companyname="Test Record",firstname=First,lastname=Last
updatecontact email=somewhere@overrainbow.com
updatecontact user1="Record added by sdOffice as a test record"
```

Sample: Outlook Email

File: s_mail.txt

```
# This example will send an email with an attachment to a dummy email
# address.  To receive an email, you should change the reference to=xxx
# to your email address.

# Start Outlook email

outlook
setfolder email

# Create a new email message.  It will be addressed to the address shown.
newemail to=trash@synergetic-data.com

# Add some information to the email.
updatemail subject=Test message
updatemail body="Attached is the sample sdOffice document template s_ltrhd.dot.\n\nEnjoy!\n"
updatemail attach=*s_ltrhd.dot

# send it
sendmail
```

Sample: Outlook Read Appointments

File: s_appt2.txt

```
# Displays appointment record(s) found on January 1, 2001.  This is the
# date of test records added by s_appt1.txt.

outlook

# set the current folder to the default appointments folder
setfolder appointments

# This sets the filter condition.  If you want to simply get every appt,
# you could do something like [subject]>"".  In addition to the
# filter, you also specify a list of fields to return.
setget [start]>="January 1, 2001 8:00 AM" and [end] <= "January 1, 2001 10:00
AM",start,duration,subject

# This command gets the first record and displays the requested fields.
# In a program, you could continue to issue getnext commands until
# just an "*" is returned, indicating no more records satisfy the
# filter criteria.
getnext

# This command returns a list of all remaining records, if any.

getall
```

Sample: Outlook Read Contacts

File: s_cont2.txt

```
# Displays the first contact record with a customer ID of "test".  The
# s_cont1.txt command file writes such a record.

outlook

# set current folder to default contacts folder
setfolder contacts

# This sets the filter condition.  If you want to simply get every name,
# you could do something like [companyname]>"".  In addition to the
# filter, you also specify a list of fields to return.

setget [customerid]="test",name,firstname,lastname,email

# This command gets the first record and displays the requested fields.
# In a program, you could continue to issue getnext commands until
# just an "*" is returned, indicating no more records satisfy the
# filter criteria.
getnext

# This command returns a list of all remaining records, if any.

getall
```

Sample: Word Document Formatting

File: s_word1.txt

```
# Writes a letter to demonstrate features of Word automation

# start Word and open a document template from the sdOffice directory.
word
newdoc *s_ltrhd.dot

# show the document as we work - note: bad for performance
show

# change default font characteristic
font bold

# Add an address
write \n\nCompany Name\nAttention: John Smith\n123 Street Address\nSuite 255
write Anytown, CA 99556\n\n

# Change the font
font name=Times New Roman,size=18,bold
paragraph center,shade=10,indent=.75,rightindent=.75
write sdOffice Example

#change the paragraph and font back to normal
paragraph normal
font normal
write \nHere is some text for a paragraph.  To continue the write command\
 end it with a backslash (\) and continue writing on the next\
 line.
write \nHere is a table:

screenupdating off - helps on performance until turned back on
# tables are defined, then written to, then formatted
table center,borders=grid,autofit
tablerow Invoice\tDate\tAmount
tablerow 12345\tOct 1,2000\t9,999.00
tablerow 34567\tNov 15, 2000\t12,121.21
tabledef row=1,backcolor=gray25
tabledef col=last,right

screenupdating on

write \n

# Do some lists
write Bullet list:\n
bulletlist Bullet list item 1\nItem 2\nItem 3\nItem 4
write \nNumber list:\n
numberlist Number list item 1\nItem 2\nItem 3\nItem 4

# add an image at certain point positions
units points
image file=*s_logo.tif,x=504,y=684,w=72,h=72
```

```
# do a print preview to show result
printpreview
leaveopen
```

Sample: Word Mail Merge

File: s_word2.txt

```
# Mail merge sample using existing Word document s_word2.doc, which
# contains mail merge tags.

# Startup Word

word

# Watch what's going on (in practice, probably shouldn't be used,
# at least until the end of the job, for performance reasons).

show

# Create a new document with letter head to store mail merge results,
# and open the base document, s_word2.doc in the sdOffice directory.
# s_word2.doc will be the active document.

newdoc *s_ltrhd.dot
opendoc *s_word2.doc

# Perform the mail merge substitutions.

mergefield CompanyName=Microsoft,Addr1=2111 Beach Blvd.,Addr2=""
mergefield City=Somewhere,State=XX,ZipCode=99999,Balance="4,500.00"

# Copy the document, activate the new document, paste, and return
# to s_word2.doc.

copydoc
activate Document1
pastedoc
activate s_word2.doc

# Perform another set of substitutions

mergefield CompanyName=Oracle,Addr1=2111 Beach Blvd.,Addr2="Suite 1"
mergefield City=Somewhere,State=XX,ZipCode=99999,Balance="88,300.00"

# Another copy, activate, paste.  This time, add a page break before
# pasting.
copydoc
activate Document1
newpage
pastedoc

# Drop the s_word2.doc (don't wan't the user accidentally saving it)
activate s_word2.doc
closedoc
activate Document1

# Show the result
```

printpreview

leaveopen

Colors in Word and Excel

sdOffice provides access to many internal color designations provided by Word and Excel. When automating those applications, you can set color and background color elements to any of the following words:

- black
- blue
- red
- green
- yellow
- white
- darkblue
- darkred
- darkyellow
- turquoise
- teal
- pink
- violet
- brightgreen
- gray25
- gray50

Excel, but not Word, also supports these colors:

- magenta
- cyan

The default color is black, and the default background color, where supported, is white.

Paper Bins

Word and Excel printer selection can include a specification of the paper bin to use. The following paper bin names are valid:

- upper
- lower
- middle
- manual
- envelope or env
- envmanual

Paper Sizes

Word and Excel page setup commands support named paper size definitions. The following list identifies the valid page size names. Any value other than these in a **papersize=*name*** option will result in selection of Letter size.

- letter (8.5x11 inches)
- legal (8.5x14 inches)
- a3 (297x420 mm)
- a4 (210x297 mm)
- a5 (148x210 mm)
- b4 (250x354 mm)
- b5 (182x257 mm)
- tabloid (11x17 inches)
- env9 (3-7/8 x 8-7/8 inches)
- env10 (4-1/8 x 9-1/2 inches)
- env11 (4-1/2 x 10-3/8 inches)
- env12 (4-1/2 x 11 inches)
- env14 (5 x 11-1/2 inches
- envmonarch (3-7/8 x 7-1/2 inches)
- envb4 (250 x 353 mm)
- envb5 (176 x 250 mm)
- envb6 (176 x 125 mm)

SendKeys Characters

The SendKeys command offered by some automation objects supports special keys and key combinations, in addition to standard text. The following special keys are recognized as names enclosed in braces:

Keystrokes
{backspace} or {bksp} or {bs}, {break}, {delete} or {del}, {enter} or ~, {esc}, {tab}

Cursor Movement
{down}, {end}, {home}, {left}, {pgdn}, {pgup}, {right}, {up}

Keyboard State
{capslock}, {insert} or {ins}, {scrolllock}

Function Keys
{F*n*}, i.e. {F10}

Escaped Characters
Since some characters have special meaning, they must be enclosed in braces:
{+}, {^}, {%}, {~}, {(}, {)}, {{}, {}}, {[}, {]}

To repeat a key, append a space and a number after any key inside braces. For example {A 10} will send 10 "A" keys, and {right 5} will send 5 right-arrows.

You can also imply a Shift, Control, or Alt combination by preceding the keystroke with +, ^, or % characters, respectively. For example, to issue an "Alt-E, C" sequence to mimic a "copy" function, use SendKeys %ec. The %e implies an Alt-E keystroke, and the letter c would select the Copy option of a normal Windows application edit menu.

Index

T

V

W