

sdOffice®

Version 2.0

Copyright ©2007 by Synergetic Data Systems Inc.
All rights reserved.

sdOffice is a registered trademark of Synergetic Data Systems Inc. Other product names used in this document may be trademarks or registered trademarks of their respective owners.

TABLE OF CONTENTS

TABLE OF CONTENTS.....	2
INTRODUCTION	4
ARCHITECTURE	5
NETWORK SERVER INSTALLATION	6
Unix/Linux Server From Download	6
Unix/Linux Server from CD	6
Windows Server.....	8
OFFICE CLIENT INSTALLATION	9
CONFIGURATION.....	10
Unix Configuration	10
Windows Configuration.....	12
LICENSING	13
Unix Licensing.....	14
Windows Licensing	15
Activation Errors.....	16
WINDOWS OFFICE CLIENT OVERVIEW	17
Overview.....	17
APPLICATION CLIENT INTERFACES	18
Pre-defined Interfaces	18
Default Rules for Client and Server Addresses	18
Command Files - sdRun Interfaces.....	20
Standard I/O - sdpipe.pl Interface.....	22
BBx and ProvideX Interfaces	23
Print Preview – sdpreview.sh.....	27
Direct Interface Guidelines	28
OBJECT REFERENCE	30
Job Management	31
Command Usage and Parameters	31
Universal Commands.....	32
Command Usage and Parameters	32
Excel Object.....	40
Overview.....	40
Command Usage and Parameters	41
Word Object.....	57
Overview.....	57
Command Usage and Parameters	58
Outlook Object.....	69
Overview.....	69
Command Usage and Parameters	69
MAPI Object.....	77

Overview.....	77
Commands and parameters.....	78
ADO Object.....	80
Overview.....	80
Command Usage and Parameters.....	81
System Object.....	90
Mail Object.....	91
Overview.....	91
Command Usage and Parameters.....	91
SAMPLES.....	95
Sample: ADO Database Manipulation.....	96
Sample: Excel Calculation Engine.....	97
Sample: Excel Formatting.....	98
Sample: Excel Report.....	99
Sample: Excel Charting.....	101
Sample: MAPI email submission.....	102
Sample: Outlook Add Appointment.....	103
Sample: Outlook Add Contact.....	104
Sample: Outlook Email.....	105
Sample: Outlook Read Appointments.....	106
Sample: Outlook Read Contacts.....	107
Sample: Word Document Formatting.....	108
Sample: Word Mail Merge.....	110
ADMIN CONNECTIONS.....	112
OBJECT EXTENSIONS WITH VBSCRIPT.....	113
AUTOMATED FILE DISTRIBUTION.....	116
APPENDIX.....	118
Colors in Word and Excel.....	118
Paper Bins.....	119
Paper Sizes.....	120

INTRODUCTION

sdOffice is a unique software tool that enables access to Microsoft Office® and other Windows-based technologies from non-Windows systems and applications. Any application or computer that can interact with a TCP/IP socket or Unix/Linux pipe can now work with Microsoft Office and other products, using sdOffice as the bridge to the Component Object Model (COM) interfaces found in many Windows applications.

COM is a method used in the Windows application world to treat programs, such as Microsoft Excel® and Microsoft Word®, as programmable objects from within other programs. This powerful facility is an important part of the Microsoft Windows® platform, but many legacy programming environments are not able to take advantage of it. In addition, many applications today run in Unix and Linux environments and have no direct way to access COM objects on Windows systems.

sdOffice is the bridge that allows applications of any sort to work with the COM object model for Microsoft Office. sdOffice accepts commands from an application and translates them to the appropriate COM methods, allowing applications written in virtually any programming language, and running on virtually any operating system, to control programs like Excel and Word in real time with a comprehensive, extensible set of commands.

As an example of the sort of capability that sdOffice provides, imagine the example of a traditional export to Excel procedure used by many legacy software users. In the original effort, the user runs a report that creates a tab-delimited or comma-separated-value text file. The user then copies the file to their PC, opens Excel, and imports the file. The columns in that file show up in the cells in Excel, and then the work begins. The user begins formatting the columns to match the data types and formats, adds column headings, inserts a formula column, adds column totals or maybe some grouped subtotals. After all this work, the Excel worksheet finally looks like a finished document.

With sdOffice, that same report that started the whole process can export directly to Excel and perform every one of those formatting, formula, and totaling steps without any user intervention. With that sort of automated functionality, the application, rather than the user, performs all the work to produce the finished document, saving a tremendous amount of time.

sdOffice is composed of several elements, which are described in the next chapter.

ARCHITECTURE

sdOffice is composed of three elements:

- An sdOffice Network Server
- Windows-based Office clients
- Application clients

The heart of sdOffice is the Network Server. This software runs full time in background, optionally as a service under Windows, and as a daemon on Unix or Linux. The Network Server is the interface point for both Office clients and Application clients.

The [Office clients](#) connect to the network server when they start up, and maintain a persistent connection while waiting for application clients to connect and initiate jobs. The Office client is the piece that translates commands into COM object methods. The Office client is similar to, but more powerful than, the old sdOffice 1.0 Windows client.

The [Application clients](#) connect to the network server whenever a job needs to be executed on an Office client. The Application connects to the Network Server, specifies which client and which job object, and then begins sending commands. The Network Server routes the commands to the appropriate client, and returns responses back. Application clients include:

- The application software itself, if it can work with a TCP/IP socket
- A Perl-based pipe for applications that can use bi-directional pipes but not sockets
- sdRun executables (Perl-based on Unix/Linux, native on Windows) which process the commands from a text file
- BBx and ProvideX wrappers around the above features, providing a familiar CALL interface to programmers in those languages

It is also possible to emulate an Application Client using Telnet, or by using the Application Connection window in the Office Client's visual interface.

NETWORK SERVER INSTALLATION

Unix/Linux Server From Download

1. Login as root.
2. Create a directory to hold the sdOffice Server files, and change to that directory.

Example: **umask 0**
 mkdir /usr/sdofc20
 cd /usr/sdofc20

3. Uncompress and extract sdOffice from the download file.

```
uncompress sdo20_xxx_tar.Z  
tar xvf sdo20_xxx_tar
```

4. Execute the set up script.

```
./sdsetup.sh
```

The `sdsetup.sh` script will create a launch script called `/usr/bin/sdo20d`, used to start and stop the sdOffice Network Server.

5. Activate demo mode, or activate permanently, using **`./license.sh`**.
6. Start the server: **`sdo20d start`**
7. Check the log file to verify operation: **`more sdo20d.log`** or **`tail sdo20d.log`**.

See the **Licensing** section for activation information.

Note that you will probably want to place the **`sdo20d start`** command in your system boot scripts, often found in the `/etc/init.d` directory or a similar location, depending on your version of UNIX.

Unix/Linux Server from CD

1. Login as root.

2. Mount the CD as a file system that supports lowercase file names. If you are unsure how to do this, check your man pages: **man mount**. The following table illustrates sample mount commands for various operating systems, assuming standard CD device names and that the mount directory /mnt is available. You may need to adjust these commands according to your configuration.

SCO UNIX OS5	mount -o lower /dev/cd0 /mnt
SCO UNIX	mount -r -f HS,lower /dev/cd0 /mnt
Unixware	mount -F cdfs -r /dev/cdrom/cdrom1 /mnt
AIX	mount -vcdafs -r /dev/cd0 /mnt
Sun Solaris	mount -rt hdfs /dev/sr0 /mnt
HP/UX	mount -r -F cdfs -o cdfcase /dev/dsk/c1d0s2 /mnt

3. Change to the sdOffice 2.0 UNIX directory in the mount directory: **cd /mnt/sdo20/unix**
4. Run the install script: **./install.sh**, or if you do not have execute permission to the file, **sh install.sh**. You must agree to the license agreement, then you will be presented a list of operating system versions. Choose the correct version for your system.
5. The sdOffice server will then be installed to the selected directory, and the set up script **./sdsetup.sh** will be automatically executed in that directory.

The **sdsetup.sh** script will create a launch script called **/usr/bin/sdo20d**, used to start and stop the sdOffice Network Server.
6. Activate demo mode, or activate permanently, using **./license.sh**.
7. Start the server: **sdo20d start**
8. Check the log file to verify operation: **more sdo20d.log** or **tail sdo20d.log**.

See the **Licensing** section for additional activation information.

Note that you will probably want to place the **sdo20d start** command in your system boot scripts, often found in the /etc/init.d directory or a similar location, depending on your version of UNIX.

Windows Server

1. From the CD, use Explorer to locate the D:\sdo20\win directory, and double-click the setup.exe program (*use Control Panel Add/Remove Programs if the system supports Terminal Services*). If you downloaded sdOffice from the Internet, simply execute the downloaded executable (*use Control Panel Add/Remove programs if the system supports Terminal Services*). Follow the on-screen prompts from the installer to install sdOffice to your system. This will install the sdo20d.exe server program, along with supporting files and the run-time engine, and create Start menu options under the title “sdOffice 2.0 Server”.
2. Click the **Server Configuration** option from the Start menu. This will conditionally rename certain files and prompt for several configuration values. The values entered are stored in the sdo20d.ini file. You can also use the **Configure** button from within the sdOffice Server Manager.
3. Click the **Server Manager** option from the Windows Start, Programs, sdOffice 2.0 Server menu.
4. Activate the demo mode, then if desired, activate permanently, by pressing the **Licensing** button and using the form that displays. On line help is available if needed.
5. Click the **Start** button from the Server Manager to start the server manually.
6. Check the log in the Server Manager to monitor operation.
7. If desired, and you are running the server on Windows NT, 2000, XP or any of the Windows variants that support NT Services, you can install the server as a service by running the **Install as a Service** option. When the sdOffice Server is run as a service, it is automatically started when Windows boots up. You must start and stop the service using the Windows Services applet, found in the Control Panel Administrative Tools option. The Server Manager options for starting and stopping the server are disabled.

See the **Licensing** section for activation information.

OFFICE CLIENT INSTALLATION

The office client is a separate installation from the server. This Windows program should be installed on any workstation that will perform sdOffice jobs. Simply run the sdOffice2_setup.exe program.

If you install it on a Windows system that supports Terminal Services, such as Windows 2003 Server, it must be installed using Control Panel, Add/Remove Programs while logged in as an administrator. This technique can be used in lieu of directly running the setup program on any system, but it is required in a server environment.

Once installed and started, choose the Options tab to configure the Office Client. See the Office Client Overview chapter for more details.

Automatic Configuration

At the end of the setup execution, the setup program will look for the file “sdOffice2.ini” in the same path as sdOffice2_setup.exe. If found, that file will be used as the default configuration file. This procedure allows an administrator to configure one Office client installation normally, then copy the sdOffice2.ini file from his/her Office Client install path to the location where the sdOffice2_setup.exe program is normally installed from (or distribute the file with the setup.exe). Users will then get a default configuration, including server and port settings, automatically.

CONFIGURATION

Unix Configuration

The network server is configured by editing the sdo20d.ini file. Here is an example file:

```
[defaults]
logfile=sdo20d.log
logdetail=1
# client security is controlled as follows:
#   secure=0   unauthenticated, open (suggested for LAN only)
#   secure=1   authenticated connections but data sent in clear (VPN
#               or low-value data traffic)
#   secure=2   authenticated, SSL encrypted
secure=0
refresh_manifiest=0

[clients]
port=6115
allow=192.*.*.*;99.99.99.99

[apps]
port=6114
allow=192.168.1.10;192.168.1.11

[admin]
#uses apps port
allow=127.0.0.1
```

The options for this file are defined below.

[defaults] section	
logfile	Connections, and optionally detail transaction data, are logged to this file.
logdetail	If set to 0, only connections are logged. If set to 1, transaction details are logged. If turned on, be sure to watch the size of the log file, and restart the Network Server periodically to reset the log.
secure	This value sets the security level for client connections, which often can come from remote systems across the Internet. There are three levels: 0 - Neither encryption nor authentication is performed on the socket. Note that all connection types are filtered by IP address to ensure only specific addresses can connect to the server. This option is

	<p>typically all that is required on a local network, as long as the data that will be sent to clients is not highly sensitive.</p> <p>1-Authentication only. No encryption is performed, but clients must authenticate themselves as valid sdOffice Office clients.</p> <p>2-Authentication and encryption. Office Clients must authenticate themselves, plus all client traffic is encrypted using SSL. If the host system does not support SSL (i.e. the openssl libraries are not available), then this option will produce a startup error and the sdOffice Network Server will not start.</p>
jobhist	<p>This value controls the creation and purging of job history files. Job history files are placed in the jobhist directory under the server install directory. Each job's commands are stored in a job history file, allowing for easy retrieval of jobs for analysis or debugging. Error messages returned by the client are entered into the job history file as a comment.</p> <p>The value is the number of days that job history files are retained. Anywhere from 0, meaning no job history files are created, to a fraction of a day or any number of days.</p> <p>Jobhist=1 would keep job history files for 24 hours. Jobhist=.04167 would keep them for 1 hour. The job history files are named</p>
refresh_manifest	<p>Sets the number of minutes between client re-processing of the manifest.txt file in the server "files" directory. Set to 0 to not refresh (the file will only be read as clients start up and connect to the server). This value should be an integer, and fractional minutes are rounded to the nearest integer. See the Automated File Distribution chapter for details.</p>
[clients] section	
port	<p>The port on which the Network Server listens for Office client connections. This defaults to port 6115. If changed, Office clients must be configured to use the new port.</p>
allow	<p>A semi-colon delimited list of valid IP addresses or address wildcards for Office clients. Clients whose IP address is not in this list are denied access.</p> <p>Common address wildcards for local networks include 192.*.*.* and 10.*.*.*.</p>
[apps] section	
port	<p>The port on which the Network Server listens for Application client connections. This defaults to port 6114. If changed, Application clients must use the new port.</p>
allow	<p>A semi-colon delimited list of valid IP addresses or address</p>

	<p>wildcards for Application clients. Clients whose IP address is not in this list are denied access.</p> <p>Common address wildcards for local networks include 192.*.*.* and 10.*.*.*.</p>
[admin] section	
allow	<p>A semi-colon delimited list of valid IP addresses or address wildcards for admin connections. Connections from IP addresses not in this list are denied access. This is often set to the localhost address, 127.0.0.1, allowing admin access only from the network server host machine.</p> <p>Admin connections are initiated via the Application client port, typically via telnet, and the first command line sent is always "admin". Admin connections provide several commands to manage the Network server.</p>

Windows Configuration

On Windows, you can edit the sd20d.ini file just as on Unix, or you can use the sdOffice Server Manager and select the Configure button. The Configuration screen provides editing capabilities to many of sdo20d.ini settings.

See the previous section for details about setting the various fields on this form, or view the on-line help.

LICENSING

Licensing for sdOffice is based upon the number of concurrent Application clients that will connect to the sdOffice Network Server. Application clients typically connect only long enough to run a job, from a few seconds to a few minutes, so the number of connection licenses required typically will be less than the number of active users on a network. The connection counts available are 1, 3, 5, 10, 15, 20, 25, 30, 40, 50, 75, 100, and unlimited.

Each sdOffice Network Server is issued a unique serial number and two activation keys. One key enables the server itself, and is tied to the server's system ID and machine class, both generated at the time of install. The other key enables the licensed number of Application connections.

Licensing is controlled entirely by the sdOffice Network Server process, sdo20d. You can install the Application interfaces and Office clients freely anywhere on your network.

Each sdOffice Network Server installation has a serial number. There is one special serial number, SD0099999, reserved for demo mode use on any machine. All permanent licenses are assigned a unique serial number and must be licensed to a single machine installation. Serial numbers and their associated PIN codes are assigned by SDSI when sdOffice is purchased. In order to obtain permanent or emergency temporary activation keys, the serial number and PIN code are required.

There are two activation keys that must be entered for full operation of sdOffice: a system key and a connections key. The system key enables the sdOffice Network Server to operate on a specific computer. The connections key determines the number of concurrent Application connections that may run. For demo mode operation, just a temporary system key is required; demo mode operation automatically enables 3 Application connections.

There are three types of system activation keys:

30-Day Demo

This license has a fixed serial number (SD0099999) and can run on any machine for 30 days. While running under this serial number, sdOffice will print “*Demo*” phrases randomly in data sent to sdOffice clients. This is the first mode activated after an installation, as it enables the retrieval of a System ID and Machine Class needed for permanent licensing later, as well as allowing sdOffice to operate in demo mode.

Permanent

This license has an assigned serial number, and requires a System ID and Machine Class to activate. A permanent license does not expire, enabling the sdOffice Network Server to run perpetually on the machine where installed and licensed. The System ID is derived from a given installation machine and attributes of a file in the sdOffice rt\lib\keys directory (Windows) or the rt\lib directory (UNIX), so it will change if the installation is moved to a new machine, or even to a new location on the same machine. Once the System ID changes, the permanent activation key

will no longer work, and sdOffice must be re-activated.

If the original permanent installation of sdOffice is no longer used, then you can request a reset of the permanent license to enable a new System ID and Machine Class to be associated with the permanent activation key. Contact sales@synergetic-data.com to request resets.

Emergency Temporary

This license is assigned a serial number, like a permanent license, but it does not require a System ID or Machine Class to activate. This allows you to re-install the sdOffice Network Server on a different machine than originally licensed, and operate it for 30 days. Once a temporary license has been issued for a given serial number, another temporary license cannot be issued for 45 days.

Unix Licensing

To activate the sdOffice Network Server on UNIX, perform the following steps:

- Login as root.
- **cd** to the sdOffice directory (i.e. `cd /usr/lib/sdsi/sdofc20`).
- Execute **./license.sh**.

The license.sh script prompts for the following options:

```
SDOFFICE LICENSING OPTIONS

Use the following options if this machine is connected to the Internet:
-----
1 - Permanent Activation (requires serial number and PIN code)
2 - Emergency Temporary Activation (also requires SN and PIN)
3 - 30-Day Demo Mode Activation

Use the following options for manual activation.  Activation keys
can be obtained from http://synergetic-data.com/sd2lic.cgi.
-----
4 - Display System ID and machine class (needed for option 5)
5 - Enter Permanent Activation
6 - Enter Emergency Temporary Activation
7 - Enter 30-Day Demo Mode Activation

q - quit
Enter selection:
```

To obtain either a permanent or emergency temporary activation, you will need to know your serial number and PIN code previously assigned by SDSI. These values are not necessary to obtain a 30-day demo mode activation.

If your machine has Internet access, you can perform activation easily by choosing options 1 through 3. Options 1 and 2 will prompt you for your serial number and PIN. Each of the three options will use the Internet to retrieve the desired activation key.

If the Internet is not available from the install machine, then you can perform activation manually by using another machine to visit <http://synergetic-data.com/sd2lic.cgi>. Use option 4 to display the System ID and Machine Class, which will be required to obtain a permanent activation key from this web site. Options 5 and 6 will prompt for a serial number, system key, and connections key, in sequence. Option 7 will only prompt for a system key.

Windows Licensing

The screenshot shows the 'sdOffice 2 Licensing' window with three main sections:

- (1) 30-Day Demo License:** Contains an 'Automatic Demo Activation' button, a text area with instructions, a 'Demo Serial Number' field (SF0099999), a 'Demo Activation Key' field, and a 'Manual Demo Activation' button.
- (2) Get System ID:** Contains a 'Show System ID' button, a 'System ID' field, and a 'Machine Class' field.
- (3) Activation:** Contains a 'Serial Number' field (<Need System ID>), an 'Auto Activate PIN' field, an 'Emergency Temp Activation' checkbox, an 'Automatic Activation' button, a 'System Key' field, a 'Connections Key' field, and a 'Manual Activation' button.

At the bottom of the window are 'Help' and 'Close' buttons.

The first step after an installation is to activate demo mode. This initializes the system ID file, enabling a permanent license to be obtained. If you get an error message after pressing the Show System ID button, then this installation has never been initialized, and you must activate demo mode first.

To activate demo mode:

If you are connected to the Internet, press the Automatic Demo Activation button. This will obtain a current demo mode activation key from SDSI's website and activate the run-time engine.

If you are not connected to the Internet, go to a computer that is, and go to <http://synergetic-data.com/sd2lic.cgi>, then click the link to get a 30-day trial. Note the activation key returned, and enter it exactly the same way in the Demo Activation Key field, then click the Manual Demo Activation button.

To verify the activation, click the Show System ID button. If the System ID and Class fields get filled in, then it worked.

To activate permanent mode:

To activate automatically over the Internet, you need to click the Show System ID button to get the System ID and Machine Class fields. Then fill in your serial number and PIN code, and click the Automatic Activation button. This will use your information to obtain a permanent activation key for the system, as well as your job and designer activation keys, and activate everything.

To activate sdOffice manually, note your System ID and Machine Class, then go to <http://synergetic-data.com/sd2lic.cgi>. Enter your serial number and PIN code, then click the button to get a permanent license. When prompted, enter the System ID and Machine Class exactly as noted on this screen. Note the two activation keys returned, and enter them exactly as provided in the two entry fields, then click the Manual Activation button.

To activate in emergency temporary mode:

To obtain a temporary activation over the Internet or manually, follow the steps for a permanent license, but check the Emergency Temp Activation option. The System ID and Machine Class are not used for temporary activations.

Activation Errors

Permanent activation keys are dependent on the system ID and machine class information generated by an installation. Therefore, a permanent activation key will only work on the original installation for which it was generated. If the sdOffice Network Server needs to be moved or re-installed, a new permanent activation key must be generated. This is only possible if SDSI resets the permanent key for your serial number, so you must contact SDSI, certify that the original installation is no longer in use, and request a reset.

In the meantime, you can obtain an emergency temporary activation to allow your serial number to be used on a new installation for 30 days.

If you attempt to get a new permanent activation key and are notified that one has already been assigned, then contact SDSI to request a reset. If this cannot be done in a timely fashion, get an emergency temporary key instead, and then contact SDSI at a later time.

Note that temporary keys are issued at most once every 45 days. If you get an error message indicating the temporary key availability has not expired, then you must contact SDSI to get a reset.

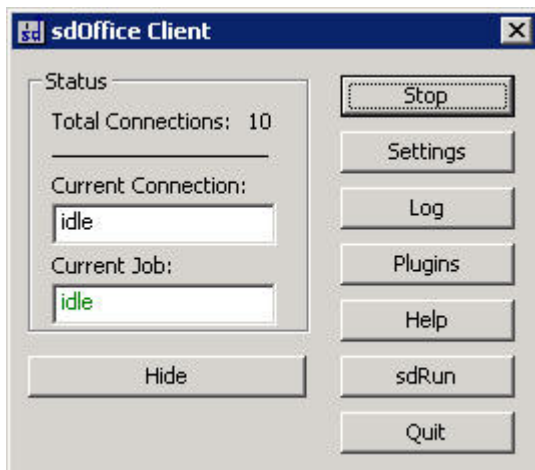
WINDOWS OFFICE CLIENT OVERVIEW

Overview

The Office Client runs on Windows-based computers and executes the jobs routed to it from the sdOffice Network Server. This program provides the interface to Microsoft Office and other objects, by translating job command streams into COM object methods and properties.

This program provides a user interface to view and edit job activity and configuration parameters. There is also a simple interface to test jobs on the client, without connecting to a Network Server and consuming application connection licenses.

For details about this interface, consult the Help menu in the Office Client.



APPLICATION CLIENT INTERFACES

The basic principle behind any application client is to connect to the sdOffice Network Server, send an initialization command, and then begin sending commands for the job. The server parses the initialization command and attempts to link the Application client with an appropriate Office client.

Pre-defined Interfaces

Applications which use the pre-defined interfaces do not need to worry about the internal send and response formats used by the server. The pre-defined interfaces are:

- sdRun interfaces (sdrun.exe on Windows and sdrun.pl (Perl-based) on Unix/Linux). These interfaces submit a job command file, which is simply a text file containing the object and the commands to send to that object.
- sdpipes.pl, a Perl-based script designed to provide a standard I/O method of interfacing with the server, for application environments that can't communicate directly with sockets.
- sdfc*.bb/sdfc*.pv, BBx and ProvideX callable programs designed for operation in those programming environments.
- sdpreview.sh, a Unix/Linux shell script designed to display a report print-preview on a user's PC workstation, simply by printing to it. The script uses the sdrun.pl Perl script, so Perl is a prerequisite.

In addition to these interfaces, any application that can communicate with the server via a TCP/IP socket (or the sdpipes.pl script) can connect and execute jobs by following the guidelines for connecting to the server and submitting jobs. These guidelines are described in the section titled Direct Interface Guidelines.

Default Rules for Client and Server Addresses

When an application wishes to start a job, it must connect to an sdOffice Network Server, and tell it what target sdOffice Client to send the job to. The application interfaces can be told explicitly what these values are, but there are other ways to specify the information.

For the Network Server, the default address is 'localhost', and the default port is 6114. In order to specify a different default, you can establish an environment variable SDSERVER with the format *server* or *server:port*. The *server* value can be a hostname or an IP address. The port portion can optionally be specified in another environment variable, SDPORT.

In BBx or ProvideX environments, when using the sdrun.bb/pv or sdfc*.bb/pv programs, you can instead specify a global string "\$sdserver", using the stbl (gbl on ProvideX) function. The port value can optionally be specified in the global string "\$sdport".

For the target sdOffice Client, the default on Unix or Linux is to evaluate the “who” command to determine the hostname of the client PC. This value will generally be accurate if the user is logged in over a telnet or ssh connection (typically via a terminal emulator). If this value isn’t valid, then an environment variable SDHOST can be defined. On Windows, there is no default value, so the target must be explicitly provided or the SDHOST environment variable must be defined. Note that in BBx or ProvideX, the stbl or gbl value “\$sdhost” can be used in lieu of setting the SDHOST environment variable.

The target can be specified in several ways: an IP address, a computer name (as the Windows PC identifies itself) in the format *@name*, or a user name (as the Windows user logged in) in the format *~name*.

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

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 Perl program:

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

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

Target is an optional argument that specifies the sdOffice Client target for the job. If not specified, the environment variable SDHOST is used, or the who command is used to attempt to identify the target PC. Note that the who command technique will not work if sdrun.pl is run in background.

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 from the environment variable SDSERVER (or localhost) and the environment variable SDPORT (or port 6114).

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:

```
\path\to\sdrun.exe CommandFile {Target {ServerPort {ServerIP {ResponseFile {ErrorFile }}}}}
```

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

Target is an optional argument that specifies the sdOffice Client target for the job. If not specified, the environment variable SDHOST is used.

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-type commands, a line "<< multiple response break >>" will appear between each result set.

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.

To execute the PRO/5 or Providex programs:

```
call "sdrun.bb|pv", CommandFile$, Target$, Server$, Response$, Errmsg$
```

CommandFile\$ is the command file path name to process.

Target\$ identifies the target sdOffice Client, by IP address, Computer Name (syntax "@name"), or User Name (syntax "~username"). If this value is null, then the default rules for determining the client are used. These rules are defined in the Application Interfaces chapter.

Server\$ identifies the server IP or hostname and optionally the port (format *server* or *server:port*) of the sdOffice Network Server. If this value is null, the default rules for determining the server are used. These rules are defined in the Application Interfaces chapter.

Response\$ will contain all the responses 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.

Standard I/O - sdpipe.pl Interface

The sdpipe.pl interface is a simple Perl script that communicates standard input and output over a socket, enabling application programs without TCP/IP socket support to open a connection to the sdOffice Network Server and execute jobs through it.

The syntax of sdpipe.pl is:

perl /path/sdpipe.pl { *server* { *port* }}

The optional arguments are *server* and *port*, which default to 'localhost' and '6114', or the values in environment variables SDHOST and SDPORT, if supplied. Once a pipe is opened to this command line, writes to the pipe are sent to the server, and reads from the pipe return server responses.

Any program that uses sdpipe.pl must adhere to the direct interface guidelines, including terminating all commands with a CR-LF sequence (chr(13) + chr(10)), and reading "get" responses until a terminating line with a single period ("."+chr(13)+chr(10)).

BBx and ProvideX Interfaces

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.

Target machines can be automatically determined by `sdofc.bb/pv` in a Unix environment where users connect via telnet or ssh, based on the ‘`who -m`’ or ‘`who -mx`’ command. However, if you are running outside of this environment, or you want to target an Office Client on a different system than the one running the terminal emulator, you need to specify the target using the string table global “`$sdhost`”. For example:

```
x$=stbl( "$sdhost", "192.168.1.20" )
x$=stbl( "$sdhost", "@mycomputer" )
x$=stbl( "$sdhost", "~JohnSmith" )
```

Under ProvideX, use `gbl()` rather than `stbl()`. Under BBx, you can also define values with “`set`” commands in the `config.bbx` file:

```
set $sdhost=192.168.1.20
```

Also note that if the sdOffice server is on a different machine than the BBx or ProvideX application, you need to define `stbl/gbl` “`$sdserver`” to specify the server machine and port:

```
x$=stbl( "$sdserver", "192.168.1.99:6115" )
```

Generic Interface Program – `sdofc.bb` or `sdofc.pv`

There is program `sdofc.bb` (and `sdofc.pv` for ProvideX) that manages the socket communication with the sdOffice server. The CALL arguments for `sdofc.bb/pv` are `object$`, `cmd$`, `response$`, `errmsg$`. This program can be used to manage a session with any sdOffice object, simply by passing the object in the first argument. For example:

```
obj$="excel"
call "sdofc.bb",obj$,"newbook",response$,errmsg$
call "sdofc.bb",obj$,"writecell col=1,row=1,value=123.45",response$,errmsg$
call "sdofc.bb",obj$,"leaveopen",,,,,""
call "sdofc.bb",obj$,"close",,,,,""
```

Object-Specific Interface Programs

For most objects, there is also a specific program that can be called that matches the object. For example, the program `sdofc_e.bb` is the BBx version of the Excel program. These object-specific programs are provided for compatibility with sdOffice 1. Each of these object-specific programs simply calls “`sdofc.bb`” or “`sdofc.pv`” with a static first argument.

These object-specific 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 direct socket 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 and issue the correct initialization command (word, excel, mapi, etc.).

A connection with the sdOffice Network Server is established via a socket or a pipe to sdpipe.pl, depending on the socket capabilities of the language. Revisions of PRO/5 or Visual PRO/5 after 2.2 support sockets with the addition of 'alias NO tcp' to the config.bbx file. ProvideX also supports sockets. Older versions of PRO/5 on Unix will use the sdpipe.pl interface, but for Visual PRO/5 on Windows, there is no support for the sdofc*.bb programs and you must use the sdrun.exe program to submit jobs as command files.

In either language, if stbl("\$sdserver") or gbl("\$sdserver") or the environment variable SDSERVER is defined, then a socket channel is opened to the specified Network Server. The format of the stbl/gbl or environment variable is: *server* or *server:port*. If the port can alternately be specified in the stbl/gbl "\$sdport" or the environment variable SDPORT.

The Network Server and port values default to 'localhost' and '6114', respectively.

sdOffice will attempt to connect the job to a workstation running the Office Client. This client is the *target*, and it can be specified as an IP address, a Windows computer name in the format @*name*, or a Windows user login in the format ~*login*. This value can be specified in the stbl/gbl "\$sdhost" or environment variable SDHOST. If it isn't specified, on Unix sdOffice will use the 'who' command to attempt to determine the hostname of the system from which a user is logged into Unix. In the case of a ProvideX user running WindX, the default value will be set to the WindX terminal's address rather than using a 'who' command.

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.

Here is a summary of each of the programs:

call "sdofc.bb|pv",object\$,cmd\$,response\$,errmsg\$

This program automates any valid sdOffice object name, as specified in the `object$` argument. This argument must be maintained in all subsequent CALL's for the same job, as the socket channel is tracked in part by its value.

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.

Print Preview – `sdpreview.sh`

This Unix shell script is designed to simplify the creation of simple text report “print previews”, where the text report will appear on the user’s Windows desktop in a scaled, zoom-able window. The script converts standard input to a preview job on the specified Office Client.

The syntax of the command is:

```
/path/sdpreview.sh { target { server { port }}}
```

Once running, all standard input to the script is routed to a work file, along with the proper commands required to implement a print preview object on the Office Client. Once the printing is complete, the job is submitted to the sdOffice Network Server using the `sdrun.pl` Perl script.

If the values for `target`, `server`, and `port` are not supplied, then the defaults as described in the `sdrun.pl` section will be used. However, since `sdpreview.sh` is typically run within a pipe, the attempt to automatically determine the client via the ‘`who`’ command will not work. Therefore, it is often important to establish the `SDHOST` environment variable as the user logs in to the Unix system.

Direct Interface Guidelines

The basics of this connection lifecycle can be demonstrated with this simple telnet-based example, with application commands indicated with >, and responses with <, for clarity:

```
> telnet localhost 6114
< sdOffice (app connection) 2.0.nn
> excel;192.168.1.10
< ok
> newbook
< ok
> setdelim |
< ok
> writerow Column 1|Column 2
< ok
> writerow John|Sally
< ok
> show
< ok
> leaveopen
< ok
> quit
```

In the above example, the application connects to port 6114 on the localhost machine. The sdOffice Network Server is listening for application connections on this machine and port. This is confirmed by the response “sdoffice (app connection) 2.0.nn” (nn is the revision number).

The first thing that the application needs to do is request an application object on a particular computer running the Office client. The request consists of a line of *object;target*. The *object* is one of the internal objects supported by sdOffice, such as Excel, Word, Outlook, or others, and the *target* is the Office client to which the job should be sent.

The *target* value can take several forms. First, it can be an IP address. Second, it can be a Window Computer Name prefixed with “@”, such as @sales or @billsdell610. Third, it can be a Windows user name prefixed with “~”, such as ~BillMiller or ~Sally Smith. When clients connect to the server, they identify themselves with their computer name (as defined under Control Panel, System, Computer Name), and the user logged in when the client connects. Note that if the client is installed to run as a service, it will by default login under the user name of “SYSTEM”.

In the above example, the initial line is “excel;192.168.1.10”, so it is asking for an Excel object on the client PC whose IP address is 192.168.1.10.

When the application sends this initial request line, the server compares connected clients with the request and matches up the application connection with a client connection that can support the object requested and matches the target name. If a client can be assigned to the application, the server sends an “ok” response. If not, the application would get an error message and be disconnected. All error messages start with the prefix “Error: “.

For example, if no client could be found, the following response would have been seen:

```
Error: No available clients support excel on 192.168.1.10
```

Once a connection is established, a job of the requested object is started on the client and is ready to start accepting commands from the application.

All commands, except those that begin with the letters “get”, are responded to with either an “ok” or an “Error: *message*” line. Any command that starts with “get” receives a stream response or an “Error: *message*” response.

All commands must end with a LF or CR+LF (chr(10) or chr(13) + chr(10)), and all responses will end with a CR+LF sequence.

A stream response is simply a series of lines terminated by a line containing a single period (plus the CR+LF). The terminating line is not to be considered part of the response, but rather just an indicator that the response is complete. Any lines within the data stream that begin with a period are converted to begin with two periods before they are returned to the application, so any application should watch for such lines and treat them as single-period lines.

OBJECT REFERENCE

The Office client supports a number of objects, most of which are related to Microsoft Office applications, though some simply provide some additional useful functionality. Each object supports a defined set of commands and options, and each object can support additional commands via an extension script, written in VBScript, that can be custom written and distributed to the Office client workstations.

Some commands are implemented universally. Once any object is initialized, these universal commands may be sent to the client and the client will act upon them regardless of which object is active. This allows some commands to be executed even if Microsoft Office is not present on the Office Client, by using the always available System object.

The standard objects provided by sdOffice are:

- Microsoft Excel
- Microsoft Word
- Microsoft Outlook
- MAPI (electronic mail via Microsoft MAPI)
- ADO (database access via ADODB/ODBC)
- Email (email send/receive via SMTP and POP)
- System (access to system folders and the Windows shell for launching tasks)
- Popup (pop up messages)
- Preview (text report print preview)

Job Management

Each connection established by an application names an object, such as “excel” or “word”, to work with. In addition, a connection can establish other objects if desired, and switch the *current object* to any that have been started. Commands that are sent always apply the current job. The job commands are as follows:

Command Usage and Parameters

JobCloseAll

v2.0

jobcloseall

Closes all open jobs.

JobEnd

v2.0

jobend *jobname*

Closes the job specified by *jobname*.

JobGet

v2.0

jobget

Returns the active job name.

JobList

v2.0

joblist

Returns a list of all open job names.

JobLogOff

v2.0

joblogoff

Ends logging of the job, formerly started with the JobLogOn command.

JobLogOn

v2.0

joblogon *logname*

Begins logging job commands and time information to the file *lognameYYYYMMDD_HHMMSS.sdoj*, stored in the JobLogs folder on the Office Client. Stored logs can be run from the Application Connection option in the Office Client, or with an [sdRun](#) interface. The JobLogOn and JobLogOff commands should bracket an entire job stream to be effective.

JobNew

v2.0

```
jobnew jobname,applicationobject
```

Creates a new job, naming it *jobname*, and starting the application object *object*. For example, 'jobnew job 1,excel'.

JobSet

v2.0

```
jobset jobname
```

Sets the active job to *jobname*.

A default job name is created when an application connection is established, based on the initial object requested. The name is “*objectjobn*”, such as “exceljob1” or “wordjob1”.

Universal Commands

The following commands are implemented at the connection level and therefore operate no matter what object is active.

Command Usage and Parameters

GetPath

v2.0

```
getpath foldername
```

Returns the client-side physical path of the named folder. All paths are returned with a terminating backslash. (i.e. c:\program files\)

Windows specific folder names	
Folder Name/Parameter	Windows Path
appdata	user application data folder
commondesktop	all users desktop folder
commonprograms	all users programs folder
commonstartmenu	all users start menu folder

commonstartup	all users startup folder
cookies	user cookies folder
desktop	user desktop folder
favorites	user favorites folder
history	user history folder
internet	user temporary internet folder
mydocuments or personal	user my documents folder
programs	user program files folder
recent	user recent files folder
sendto	user send to folder
startmenu	user start menu folder
startup	user startup folder
templates	user templates folder
temp	user temporary files folder
sdOffice specific folder names	
sdoffice	sdOffice client install path
samples	sdOffice samples folder
joblogs	sdOffice joblogs folder
plugins	sdOffice plugins folder

GetFile

v2.0

getfile *parameters*

Used to return the contents of a text file on the client workstation.

GetFile *options*

The *options* can include:

file=local full path-filename

This option uses a full path and filename for the desired file. If you are using the *folder* option below, only enter the filename (with extension) and not the path.

file option only example:

```
getfile file=c:\program files\sample.txt
Results = c:\program files\sample.txt
```

file option with *folder* option example:

```
getfile file=sample.txt,folder=programs
Results = C:\Program Files\sample.txt
```

file option with full path-filename and *folder* option example:

```
Invalid command syntax
getfile file=c:\program files\sample.txt,folder=programs
Results = C:\Program Files\C:\Program Files\sample.txt
```

folder=foldername

Read the file from a local folder. Valid folder names are documented in the [GetPath](#) command.

The *folder* option will concatenate the *folder* value returned with the *file* option value. So if you give a full path-filename for the *file* option and use the *folder* option, the resulting file name will cause an 'Invalid filename' error.

Popup

v2.0

popup *parameters*

The popup displays a popup message on the Office Client desktop, similar to the message popups in instant messaging applications or Microsoft Office 2003. The object accepts a number of parameters to indicate text, links, buttons, and colors.

The popup object provides two text regions, a standard text region, normally in the center of the popup, and a link text region, normally the top of the object. The link text can hyperlink to a file or URL, so if the user clicks it, the file is opened or the URL is opened in the browser.

Popup options

The *options* can include:

File=filename or url

This is a file or URL to open when the link text is clicked by the user. If the style is “office”, then it is opened when the user clicks anywhere in the popup.

Text=display text

LinkText=link text

The link text associated with the File value

Style = messenger | office

This setting controls the style of the popup window. The default style is “messenger”.

DisplayStyle = fade | scroll

This setting controls how the popup closes, either by fading to clear or scrolling to nothing.

TextAlign=alignment

bc|bottomcenter
bl|bottomleft
br|bottomright
mc|middlecenter
ml|middleleft
mr|middleright
tc|topcenter
tl|topleft
tr|topright
t|title

LinkTextAlign=alignment

bc|bottomcenter
bl|bottomleft
br|bottomright
mc|middlecenter
ml|middleleft
mr|middleright
tc|topcenter

t|topleft
tr|topright
t|title

ShowLink = true|false

The default value is true, but if set to false, then hyperlinks are not displayed as underlined links.

ShowClose = true|false

Forces the display of a close box on the popup. The default is off if the style is “messenger”, and true if the style is “office”.

Timeout = *milliseconds*

After *milliseconds* (1000=1 second), close the popup. Set *milliseconds* to 0 to force the popup to stay open until the user closes it. The default value is 0.

BackColor|bc = *colorname*

Sets the background color of the popup to the specified *colorname*. Valid colors include:

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

ForeColor|tc = *colorname*

Sets the color for text on the popup. Valid color names are the same as indicated in the BackColor option.

BackgroundColorGradient|bcg = *colorname*

Sets the gradient color, used along with the standard background color to generate a color gradient. Valid color names are the same as indicated in the BackColor option.

BackgroundStyle|bgs = *style*

Sets the background color gradient style. Can be one of:
Angled, Horizontal, Solid, Vertical

Preview_On, Preview_Off

v2.0

preview_on

preview_off

These commands turn on and off preview mode, respectively. When preview mode is on, all lines sent to the Office Client are captured in a buffer, and when preview mode is turned off, a print preview window is presented on the workstation. The contents of the print preview are

scaled based on the columns and rows received, with form feed characters (chr(12)) delimiting pages. In addition, the user can zoom, scroll, and scale the preview.

PreviewLoad

v2.0

```
previewload fullpath-filename
```

The PreviewLoad command loads a text file from a given path and displays its contents in the print preview window.

Preview

v2.0

```
preview stringdata
```

The preview command can be used to initiate a report preview window with a single string. This differs from the preview_on and preview_off commands, which capture data sent to the client and then initiate a preview window with the captured data.

GetSvrFile

v2.0

```
getsvrfile parameters
```

GetSvrFile is used to transfer a file from the server to the client.

GetSvrFile options

The *options* can include:

```
serverfile|sfile=server file
```

All server files are stored in the “files” subdirectory under the sdOffice server install path. There is no need to include path information, as just the base file name is used.

```
clientfile|cfile=client file (full path-filename)
```

This is the file name the server file is copied to. It can be a full path, or you can supply a base name and a folder name. The client file can contain references to special folders in brackets, such as “[mydocuments]specials.doc”. See [GetPath](#) for a list of special folder names.

```
folder=foldername
```

Save the client file to a folder. Valid folder names are documented in the [GetPath](#) command.

SendEmail

v2.0

sendemail *parameters*

Sends an email using SMTP. This command accepts many options.

Sendemail *options*

to|rec|recipient|sendto=*email* [,*email* ,...]

Any “to” addresses, separated by commas or entered as multiple to= options.

from|sendfrom|sender=*email*

“From” email address.

cc= *email* [,*email* ,...]

Any “cc” addresses, separated by commas or entered as multiple cc= options.

bcc= *email* [,*email* ,...]

Any “bcc” addresses, separated by commas or entered as multiple bcc= options.

subject|sub=*subject text*

body|text|bodytext=*message body text*

bodyhtml=*HTML message body*

attach=*file* [,*file* ,...]

Any file to attach to the email. Files must reside on the client workstation, not the sdOffice server. To attach multiple files, separate them with commas, or use multiple attach options.

SetFile

v2.0

setfile *parameters*

Used to write a string to a text file on the client workstation.

SetFile *options*

Valid *options* include:

text | data=*string*

The text to be written to the file. Note this overwrites the current contents.

file | filename | path=*file name (full path-filename)*

The full path and file name of the client-side file to write.

SetSvrFile

v2.0

setsvrfile *parameters*

SetSvrFile is used to transfer a file from the client to the server. All files sent to the server are written to the “files” subdirectory under the sdOffice server install path.

SetSvrFile options

The *options* can include:

Clientfile|cfile=*client file (fill path-filename)*

The client-side file name, which can be a full path or a base name located in the folder specified by the folder= option.

Serverfile|sfile=*server file*

The server-side file name. Paths are ignored, and the base name is used as a file written to the servers’ “files” directory.

Folder=*foldername*

Read the client file from a folder. Valid folder names are documented in the [GetPath](#) command.

Excel Object

Overview

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 workbook (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 workbook is automatically activated. When you create a new sheet in the workbook, it is automatically activated. You can also manually activate a workbook (ActivateBook), and a sheet within the workbook (ActivateSheet), using their names. Sheets can also be cleared of their contents, or deleted entirely (ClearSheet, DelSheet). The name of the current workbook 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. Formatting includes options to lock cells or hide cell formulas when used in conjunction with workbook or worksheet protect commands.

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.

New commands for Excel in Version 2 include: footer, getfooter, getheader, header, and sendto.

Command Usage and Parameters

Activate

v1.0

activate *workbook*

activatebook *workbook*

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

ActivateSheet

v1.0

activatesheet *worksheet*

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

AddChart

v1.0

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.

Borders

v2.0

borders *parameters*

Adds borders to the cell range specified, or the last cell range specified in a writecell command.

col=columns, identifies a column (or in combination with row, a cell) to apply borders to.

grid (boolean) – applies a grid layout to the cell range, so that there is an outer border and internal grid lines, using the attributes specified (linestyle, weight, and color) or Excel defaults.

left, right, top, bottom (booleans) - select one border side and applies the attributes specified (linestyle, weight, and color) to it, or uses Excel defaults.

linestyle=style, which can be one of:

continuous, dash, dashdot, dashdotdot, dot, double, slantdashdot, or none

range=range, identifies a cell range, such as A2:D16, to apply borders to.

row=rows, identifies a row (or in combination with col, a cell) to apply borders to.

weight=*thickness*, which can be one of:
hairline, thin, medium, thick

color=*colorname*, which can be one of:
black, blue, red, green, cyan, magenta, yellow, white, darkblue, darkred,
darkyellow, turquoise, teal, pink, violet, brightgreen, gray25, gray50

colorindex=automatic or none

ClearSheet

v1.0

clearsheet

Removes data and formatting from the current sheet.

CloseBook

v1.0

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

v1.0

delcol *columnnumber*

Deletes the column number in specified as the parameter.

DeleteDoc

v2.0

deletedoc *filename*

Deletes a file from the workstation's file system. The file must have a .xls extension.

DelRow

v1.0

delrow *rownumber*

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

DelSheet

v1.0

delsheet

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

EditChart

v1.0

`editchart` *parameters*

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 contiguous 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 interprets 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.

Footer

v2.0

footer *parameters*

Sets the print footer based on the following parameters:

- lh | leftfooter=*left footer text*
- ch | centerfooter=*center footer text*
- rh | rightfooter=*right footer text*

Format

v1.0

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*
hide (boolean)
left (boolean)
lock (boolean)
mergework (boolean)
numberformat=*format*
range=*range*
right (boolean)
row=*row*
width=*measure*
wraptext (boolean)

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 specified (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.

Lock and hide options do not take effect unless the workbook or worksheet is protected with the `protectbook` or `protectsheet` commands.

Formats

v1.0

`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

v1.0

`getbook`

Returns the name of the current workbook.

GetBooks

v1.0

`getbooks`

Returns all workbook 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, an asterisk is returned.

GetData

v1.0

`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. A range can be specified using Excel's standard format, such as `A1:B32`.

`Col=column`

`Row=row`

`Range=range`

GetFooter

v1.0

`getfooter segment`

Returns the text for a given footer *segment*, where *segment* can be one of the following:

lf or leftfooter
cf or centerfooter
rf or rightfooter

GetHeader

v2.0

getheader *segment*

Returns the text for a given header *segment*, where *segment* can be one of the following:

lh or leftheader
ch or centerheader
rh or rightheader

GetLastCol

v2.0

getlastcol

Returns the highest column number that contains data in the current worksheet.

GetLastRow

v2.0

getlastrow

Returns the highest row number that contains data in the current worksheet.

GetRow

v1.0

getrow

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

GetRows

v1.0

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**. Use the **getlastrow** command to return the highest row that contains data in the worksheet.

GetSheet

v1.0

getsheet

Returns the name of the current worksheet.

GetSheets

v1.0

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 workbook is open, or no sheets are in the active workbook, an asterisk is returned.

Header

v2.0

header *parameters*

Sets the print header based on the following parameters:

lh | leftheader=*left header text*
ch | centerheader=*center header text*
rh | rightheader=*right header text*

Hide

v1.0

hide

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

InsertCol

v1.0

insertcol *colnumber*

Inserts a new column at the column number parameter.

InsertRow

v1.0

insertrow *rownumber*

Inserts a new row at the position in the active worksheet supplied by the *rownum* parameter. The new row becomes the current row.

LeaveOpen

v1.0

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

v1.0

`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

v1.0

`newbook`

Creates a new workbook. The name of the workbook is supplied later in a [SaveAs](#) command.

NewSheet

v1.0

`newsheet` *sheetname*

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

OpenBook

v1.0

`openbook` *workbook*

Opens an Excel (.xls/ workbook) file supplied by the *workbook* parameter. The *workbook* parameter normally requires a fullpath-filename, but, by placing an asterisk (*) at the beginning of the file name, the asterisk will be substituted with the sdOffice application path. For example, *salestable.xls would find the file in the sdOffice directory.

PageSetup

v1.0

`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)
FitHeight (boolean)
Zoom (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

v1.0

`print`

Prints the current sheet.

Printer

v1.0

`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

v1.0

`printpreview`

Launches the Print Preview screen in Excel.

ProtectBook

v2.0

`protectbook password=password`

ProtectSheet

v2.0

ProtectSheet *password=password*

Applies protection to any cells formatted with locking or formula hiding using the Format command. If a password is supplied, then the password is required in order to un-protect the workbook or worksheet with Excel. See also the [UnProtectBook](#) and [UnProtectSheet](#) commands.

This command is only supported with Excel XP and above.

Run

v1.0

run *macroname* [, *arg1* , *arg2* , ...]

Runs the Public Sub-style VBA macro named in the *macroname*. Up to 30 arguments can be passed.

Save

v1.0

save

Saves the active workbook.

SaveAs

v1.0

saveas *workbook* or *options*

Saves the active workbook as the workbook name supplied in the *workbook* parameter, or based on a set of *options*. An asterisk (*) at the beginning of the file name will be substituted with the sdOffice path. For example, *SalesTable would save the workbook in the sdOffice directory as SalesTable.xls.

In addition to the simple *workbook* parameter, you can specify parameters as a series of options:

File|filename = *filename*

Fileformat = *format*

Valid formats include addin, csv, csvmac, csvmsdos, csvwindows, currentplatoformtext, dbf2, dbf3, dbf4, dif, excel2, excel2fareast, excel3, excel4, excel5, excel6, excel7, excel9597, html, intladdin, intlmacro, sylk, template, textmac, textmsdos, textwindows, unicodetext, webarchive, wj2wd1, wj3, wj3fj3, wk1, wk1all, wk1fmt, wk3, wk4, wks, workbooknormal, works2fareast, wq1, xmlspreadsheet

Password = *password*

Writerspassword=*password*

Readonlyrecommended=true|false

Createbackup=true|false

Accessmode = exclusive|nochange|shared

Conflictresolution = userresolution|localsessionchanges|othersessionchanges

Addtomru = true|false

Local=true|false

Overwrite=true|false

ScreenUpdating

v1.0

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

v1.0

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.

Note that SendKeys can be difficult to make work correctly if keystrokes cause dialogs to open up, and also different versions of Excel can behave differently to keystroke sequences.

SendTo

v2.0

sendto *recipient-email, subject, return-receipt*

Opens an Excel email dialog with the optional *recipient-email, subject,* and *return-receipt* options filled in. Excel will automatically attach the active workbook to the email.

SetChart

v1.0

setchart *chartnumber*

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

SetDelim

v1.0

setdelim *delimiter*

Sets the delimiter used by the [WriteRow](#) command to the text value of *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

v1.0

setrow *rownum*

Sets the current row, used by [WriteRow](#), to a new *rownum* value. When a sheet is created or activated, the current row is set to 1.

Show

v1.0

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

v1.0

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 *parameters* 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

v1.0

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

v1.0

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.

UnProtectBook

v2.0

unprotectbook *password=password*

UnProtectSheet

v2.0

unprotectsheet *password=password*

Removes protection from the workbook or worksheet, allowing editing of protected cells and viewing of cell formulas. See also the [ProtectBook](#) and [ProtectSheet](#) commands.

This command is only supported with Excel XP and above.

WriteCell

v1.0

writecell *parameters*

This will write a value (number, date, text or formula=*expression*) specified by *parameters*. 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

v1.0

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.

Word Object

Overview

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.

New commands in Version 2 include: sendto.

Command Usage and Parameters

Activate

v1.0

`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

v1.0

`bulletlist list`

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

ClearDoc

v1.0

`cleardoc`

Clears all the text from the current document.

CloseDoc

v1.0

`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

v1.0

`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

v1.0

`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

v1.0

getdoc

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

GetDocs

v1.0

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, an asterisk is returned rather than one or more names.

GetFields

v1.0

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

v1.0

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

v1.0

getparagraphs

Returns the number of paragraphs in the document.

GetText

v1.0

gettext

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

Hide

v1.0

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

v1.0

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=filename
h=measure
inline (boolean)
stretch (boolean)
w=measure
x=measure
y=measure

The *filename* 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

v1.0

`leaveopen`

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

MergeField

v1.0

`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

v1.0

`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

v1.0

`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

v1.0

`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

v1.0

`opendoc` *documentname or 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

v1.0

`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

v1.0

`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*
Kepttogether (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

v1.0

`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

v1.0

`pastetext`

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

Print

v1.0

`print`

Print the current document.

Printer

v1.0

`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

v1.0

`printpreview`

Executes a Print Preview of the current document in Word.

Replace

v1.0

`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

v1.0

`run macroname[arg1,arg2,...]`

Runs the Public Sub-style VBA macro named as the parameter. Up to 30 arguments can be passed.

Save

v1.0

`save`

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

SaveAs

v1.0

`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

v1.0

`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

v1.0

`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..

Note that SendKeys can be difficult to make work correctly if keystrokes cause dialogs to open up, and also different versions of Excel can behave differently to keystroke sequences.

SendTo

v2.0

`sendto`

Opens the Word email dialog to send the current Word document as an attachment.

SetDelim

v1.0

`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

v1.0

`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 artificially high number.

Show

v1.0

`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

v1.0

`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

v1.0

`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=colormame`
Center (boolean)
`Col=column|last`
`Color=colormame`
`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

v1.0

`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

v1.0

`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/Type

v1.0

`write text`

`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.

Outlook Object

Overview

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 `Newitem` command, such 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 `Edititem` commands, such as `EditAppointment`. To delete a record, use the `Delitem` commands, such as `DelContact`.

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

Command Usage and Parameters

DelAppointment

v1.0

`delappointment`

`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

v1.0

`delcontact`

Deletes the current contact record.

DelTask

v1.0

`deltask`

Deletes the current task record.

GetFolder

v1.0

`getfolder`

Returns the active folder and its type in the format “*name (type)*”.

GetFolders

v1.0

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

v1.0

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

v1.0

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 field.

LeaveClose

v1.0

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

v1.0

newappointment *parameters*

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 UpdateAppointment commands can be used to update the same record.

NewContact

v1.0

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

v1.0

newmail *parameters*

email *parameters*

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.

NewTask

v1.0

newtask *parameters*

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

SendMail

v1.0

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

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

v1.0

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
fileas
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
jobtitle
notes (or body)
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

v1.0

UpdateAppointment *parameters*

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

v1.0

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
- Any user-defined field name

UpdateMail

v1.0

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

v1.0

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=value

body=value

reminder=true|false

complete=true|false

status=complete | deferred | inprogress | notstarted | waiting

actualwork=number

totalwork=number

percentcomplete=number, 0 to 100

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

MAPI Object

Overview

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.

An alternative to the MAPI object is to use the internal “sentto” commands found in the Excel and Word objects, or the email commands available in the new System object, which use the industry standard SMTP and POP protocols for sending and receiving email, respectively.

Commands and parameters

NewMail

v1.0

`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.

SendMail

v1.0

`sendmail parameters`

`send 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

v1.0

`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

v1.0

`signoff`

Ends the current Outlook session.

UpdateMail

v1.0

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.

ADO Object

Overview

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 some 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 added in Version 2 include: addnew, delete, export, filter, getconnectionstring, getdelimiters, getmove, getmovefirst, getmovelast, getmovenext, getmoveprevious, getposition, getrecordcount, loadrs, move, movefirst, movelast, movenext, moveprevious, requery, savers, setdelimiters, sort, and update.

Command Usage and Parameters

AddNew

v2.0

`addnew options`

Adds a new record to the active recordset. The format of *options* is simply a comma-separated list of field names assigned to values. For example: `id="1234",name="Acme Rents",sales=1000`. The added record is updated in the database.

Close

v1.0

`close`

Closes the open connection.

CloseRS

v1.0

`closers`

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

Connect

v1.0

`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 administrator 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
```

See the **GetConnectionString** for information about getting the connection string via the Windows user interface.

Delete

v1.0

`delete`

Deletes the current record from the recordset and the database.

Execute

v1.0

`execute sqlstatement`

Executes the SQL statement given. If the statement 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.

ExecuteCmd

v2.0

`executecmd adocommand`

Executes an ADO Command. ADO commands are useful when you need to run a stored procedure or return a single value from a database. ADO commands are beyond the scope of this document. If you need detailed information on ado commands and their structure, please refer to the Microsoft Developers Network at <http://msdn.microsoft.com>

`commandtext` | `text` = *SQL Statement*

This is the sql statement to be run against your database.

`commandtype` | `type` = *commandtype*

The value can be any of the following literal values

file

storedproc

table

tabledirect

text

unknown (default)

`parameter` | `param` = "*parameter*"

Parameters strings must be in quotes

`commandexecute` | `execute`|`executeoption` |`option`

`timeout` = *value* (in seconds)

(default = 60 seconds)

`cursorlocation` | `location` = *cursorlocation*

client | **useclient** (default)

server | useserver

`cursorstype` | `type` = *cursorstype*

opendynamic | dynamic

openforwardonly | forwardonly

openkeyset | **keyset** (default)

openstatic | static

locktype | lock
lockbatchoptimistic | batchoptimistic
lockoptimistic | optimistic (default)
lockpessimistic | pessimistic
lockreadonly | readonly

cachysize | cache = value
(default = 500)

Export

v2.0

export *options*

Exports the current recordset to a file, based on the following options:

fp | file | filepath | filename = *filename*
Exports to the specified file name.

overwrite=yes|no|true|false
If set to yes or true, will overwrite the file specified if it already exists. Otherwise, an error message is returned.

fd | field | fielddelimiter = *value*
The *value* can be any of these literal values:

tab | \t | chr(9) for a tab
sc | semicolon | chr(59) for a semicolon
comma | chr(44) for a comma
space | chr(32) for a space

Any other value is interpreted as a literal delimiter value.

tq | text | textqualifier = *value*
The text qualifier is used to surround values that are of a text data type. This format is often required for formats such as CSV. The *value* can be one of the following literal values:

none for no text qualifier
' | squote | chr(39) for an apostrophe (single quote)
" | quote | dquote | chr(34) for a double quote (literal quote must be entered as "\"")

ld | line | linedelimiter = *value*
The *value* can be any of these literal values:

crlf | chr(10)+chr(13) for a CR-LF
lf | chr(10) for a LF
cr | chr(13) for a CR

Any other value is interpreted as a literal delimiter value.

The default values for delimiters are specified by the **SetDelimiters** command, which in turn default to settings for a comma-separated-value (CSV) file format.

Filter

v2.0

```
filter filterexpression
```

Applies the *filter expression* to the active recordset, limiting records returned from the set to those that match the filter expression. The expression syntax matches that of an SQL where clause. For example, “country = 'USA'” or “amount >=10000”. The filter expression can be set to "" to stop filtering records.

GetCols

v1.0

```
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

v1.0

```
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.

GetConnectionString

v2.0

```
getconnectionstring
```

Prompts the user to select a data source via a Windows dialog, and returns the resulting connection string. The connection string can be used in a **Connect** command.

GetDelimiters

v2.0

`getdelimiters delimitername`

Returns the current delimiter specified by *delimiter name*. The name can be one of the following:

fd | field | fielddelimiter for the field delimiter

tq | text | textqualifier for the text qualifier

ld | line | linedelimiter for the line delimiter

GetExecute

v1.0

`getexecute SQLStatement`

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.

GetMove

v2.0

`getmove options`

Similar to the **Move** command, but returns the selected row's fields delimited by the current field delimiter.

GetMoveFirst, GetMoveLast, GetMoveNext, GetMovePrevious

v2.0

`getmovefirst`

`getmoveleast`

`getmovenext`

`getmoveprevious`

Similar to the **MoveFirst**, **MoveLast**, **MoveNext**, and **MovePrevious** commands, but returns the selected row's fields delimited by the current field delimiter.

GetPosition

v2.0

`getposition`

Returns "Record *n* of *totalrecords*", indicating the current record position in the active recordset.

GetRecordCount

v2.0

`getrecordcount`

Returns the number of records in the active recordset.

GetRow

v1.0

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

v1.0

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

v1.0

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.

LoadRS

v2.0

loadrs *filename*

Loads a previously stored ADO recordset from the specified file. The stored recordset can be one created by the **SaveRS** command, or it can be one saved by another application.

Move

v2.0

move *options*

Moves the recordset cursor a specified number of records from a specified location. The options supported are:

records | numrecs = *value*

The number of records to move the cursor.

start=current | first | last

The starting to point from which to move.

MoveFirst, MoveLast, MoveNext, MovePrevious

v2.0

movefirst

movelast

movenext

moveprevious

These commands move the recordset cursor to the first, last, next, or previous record, respectively.

ReQuery

v2.0

requery

Re-executes the last query run by the **Execute** command, and resets the filter and cursor.

SaveRS

v2.0

savers *options*

Saves the recordset to a file in one of two formats. The resulting file can be loaded at a later time using the **LoadRS** command. The options are:

path | file | filename = *filename*

Saves the recordset to the specified file name.

format = xml | ado | adtg

Stores the recordset in either XML or internal ADTG format. XML recordsets are text-based and occupy more disk space. ADTG is a binary ADO format that is more compact.

SetDelim

v1.0

setdelim *delimiter*

Sets the field 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.

SetDelimiters

v2.0

setdelimiters *settings*

Sets field, text, and line delimiters to any of the following settings:

fd | field | fielddelimiter = *value*

The *value* can be any of these literal values:

tab | \t | chr(9) for a tab
sc | semicolon | chr(59) for a semicolon
comma | chr(44) for a comma
space | chr(32) for a space

Any other value is interpreted as a literal delimiter value.

tq | text | textqualifier = *value*

The text qualifier is used to surround values that are of a text data type. This format is often required for formats such as CSV. The *value* can be one of the following literal values:

none for no text qualifier
' | quote | chr(39) for an apostrophe (single quote)
" | quote | dquote | chr(34) for a double quote (literal quote must be entered as "\"")

ld | line | linedelimiter = *value*

The *value* can be any of these literal values:

crlf | chr(10)+chr(13) for a CR-LF
lf | chr(10) for a LF
cr | chr(13) for a CR

Any other value is interpreted as a literal delimiter value.

Sort

v2.0

sort *fields*

Sorts the active recordset based on the specifications in the *fields* string. The string is a comma-separated list of field names, each optionally suffixed with a keyword ASCENDING or DESCENDING. For example, “salesperson, ytd_sales DESCENDING”.

Timeout

v2.0

timeout *seconds*

Sets the timeout value for the database connection. Operations that take longer than this value will generate an error. The default value is 30 seconds. Setting *seconds* to 0 will disable timeout errors, perhaps allowing an operation to wait indefinitely.

Update

v2.0

update *options*

Updates fields in the current record. The format of *options* is simply a comma-separated list of field names assigned to values. For example: id="1234",name="Acme Rents",sales=1000. The record is updated in the database.

System Object

The system object provides access to any of the universal commands, including:

- GetPath
- GetTextFile
- PopUp
- Preview, Preview_On, Preview_Off
- GetSvrFile
- SendEmail
- SetFile
- SetSvrFile

Each of these commands is documented in the [Universal Commands](#) chapter. In cases where there is no Microsoft Office on a workstation, or if the overhead of an Office object is not needed, the System object can be used instead.

Mail Object

Overview

The Mail object provides access to SMTP and POP mail servers in order to send and receive email from an application connection. This capability is useful in cases where a Unix server doesn't have access to the Internet, and therefore the site's mail servers, but a local PC running the Office Client does.

The Mail object can send text or HTML-formatted email, along with attachments, via any SMTP server available to the Office Client. It can also check to email to retrieve and return headers or full messages, enabling development of applications that need to monitor an email account for activity.

Command Usage and Parameters

GetHeader

v2.0

`getheader options`

Returns a specific header of a given message from the POP server. The *options* available are:

ID | MsgID | MessageID = *messageno*

The message number of the message to return. Messages on the POP server are numbered from 1 to the number of available messages. The number of available messages is returned by the **GetMessageCount** command.

Header = *name*

The name of the header, such as "Subject", "To", or "From", to return.

GetHeaders

v2.0

`getheaders messagenumber`

Returns all the headers of the specified message.

ID | MsgID | MessageID = *messageno*

The message number of the message to return. Messages on the POP server are numbered from 1 to the number of available messages. The number of available messages is returned by the **GetMessageCount** command.

GetMessage

v2.0

getmessage *messageno*

Returns the entire contents of the specified message. Note that this content will contain headers followed by a blank line, and then the message contents, possibly in multiple MIME-encoded parts. Decoding of the contents, if necessary, is not a service provided by sdOffice.

ID | MsgID | MessageID = *messageno*

The message number of the message to return. Messages on the POP server are numbered from 1 to the number of available messages. The number of available messages is returned by the **GetMessageCount** command.

GetMessageCount

v2.0

getmessagecount

Returns the number of messages available at the POP server.

GetServer

v2.0

getserver

Returns the server information used by the Mail object for SMTP and POP logins. The information is returned in the following format:

```
smtpserver:value
smtpport:value
popserver:value
popport:value
username:value
password:value
timeout:value
```

SendMessage | SendEmail *options*

Sends an email message based on the following options:

To = *recipient* [, *recipient 2*] [, *recipient 3...*]

Sets the primary recipient, which may be a single email address, or a comma-separated list of addresses.

From = *from address*

Sets the From: address of the email.

Cc = *recipient* [, *recipient 2*] [, *recipient 3...*]

Sets additional recipients who will appear in the Cc: header. This can be a comma-separated list of addresses, like To.

Bcc = *recipient [, recipient 2] [, recipient 3...]*

Sets additional recipients who will not appear in the header. This can be a comma-separated list of addresses, like To.

Subject = *subject*

Sets the message subject.

Body | text = *message text*

Sets the plain text portion of the message.

BodyHTML = *html text*

Sets an optional HTML text portion of the message.

Attach = *file 1 [, file 2] [,file 3] ...*

Adds the named file(s) as attachments to the email. Note that these files must reside on the Office Client, not the sdOffice Network Server.

SetServer

v2.0

setserver parameters

Sets the server options to be used by the commands for sending and receiving email. The default values can be configured by the local Office Clients, but this method allows the application jobs to control the server environment.

SmtpServer = *server*

Sets the SMTP server's IP address or hostname.

SmtpPort = *port*

Sets the SMTP server port, which default to 25.

PopServer = *server*

Sets the POP server's IP address or hostname

PopPort = *port*

Sets the POP server port, which defaults to 110.

UserName = *login*

Sets the login to be used for POP access, and possibly for SMTP authentication. This value is often, but not always, a user email address. The mail server administrator or Internet Service Provider provides this information.

Password = *password*

Sets the password associated with the above login.

TimeOut = *seconds*

Sets the number of seconds the object will wait when connecting to the POP or SMTP server.

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\sdooffice" (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 capabilities 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\tTABLE 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\tTABLE 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_mapil.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=allenn@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_mapil.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.
updateemail subject=Test message
updateemail body="Attached is the sample sdOffice document template
s_ltrhd.dot.\n\nEnjoy!\n"
updateemail 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 want the user accidentally saving it)
activate s_word2.doc
closedoc
activate Document1

# Show the result
```

printpreview

leaveopen

ADMIN CONNECTIONS

Admin connections are used to perform administration tasks with the sdOffice Network Server.

Admin connections are accessed via the application port, which defaults to 6114. They are typically accessed via telnet, and are by default restricted to the localhost address (127.0.0.1). Once the connection is established and the server responds with “sdOffice (app connection) 2.0.nn”, enter the command “admin”. A response of “ok” indicates you can enter administration commands.

At any time, you can enter a question mark (?) to view a list of valid administration commands. They are listed in the following table:

clients [<i>match</i>]	List active clients. The optional <i>match</i> text is a simple text filter which limits the clients listed to those containing <i>match</i> . The lines include a client handle, supported object list, client user name, client computer name, client IP address, and an application handle, if an application is connected to this client.
apps [<i>match</i>]	List active application connections. The optional <i>match</i> text is a simple text filter which limits the connections listed to those containing <i>match</i> . The lines include an application handle, application name, target machine, and client handle.
page <i>n</i>	Set pager size to <i>n</i> lines (0=continuous). The default value is 23.
close app <i>n ip all</i>	Close an application handle, ip, or all.
close client <i>n ip all</i>	Close a client handle, ip, or all.
version	Show version number.
broadcast <i>popup options</i>	Show a popup window on all connected workstations, using the same command line options as the PopUp object, described in the Universal Commands chapter.
Quit	Exit the administration session.

OBJECT EXTENSIONS WITH VBSCRIPT

sdOffice objects can be extended with custom VBScript functions placed in files associated with each object. For example, the Excel object looks for VBScript functions in the file `scripts\sdoexcel.sdos` in the Office client's installation directory. A .sdos (sdOffice Script file) file is a text file containing one or more function definitions. It is in fact a VBScript module file. Whenever a job is started and an object created, the associated script file is loaded into a VBScript control and its functions are available.

Whenever a command is sent to the Office client for parsing and processing, it is evaluated to determine if it is a recognized command. If so, it is processed internally by the Office client. If it is not recognized, then an attempt is made to execute a function in the script file of the same name.

For example, you could program a function in `sdoexcel.sdos` called "custom_archive". When an application sends a command "custom_archive *parameters*" to the Office client, the function called "custom_archive" in the script file will be executed, passed the command line parameters as an array.

The way the parameters of the command line are passed is via a two-dimensional array. For example, assume the following command was issued:

```
custom_archive name="test archive", category="accounting"
```

The custom_archive function could process the arguments like this:

```
function custom_archive(parameters)

    dim i, name, value

    for i = 0 to ubound(parameters,1)

        name=parameters(i,0)
        value=parameters(i,1)

        select case name

            case "name"
                'do some code
            case "category"
                'do some code

        end select

    end function
```

In addition to the parameters provided to each function, most of the scripts get a copy of the main object being handled by the job as a module level variable. This variable can be used to read and manipulate the object itself to perform functions that are not provided in sdOffice directly. The variable names and types are as follows:

Script		Object Variable(s)	Object Variable Type
sdoado.sdos	ADO	connection	ADO Connection object
sdoexcel.sdos	Excel	excel	Excel.Application object
sdomail.sdos	SMTP/POP	n/a	None
sdomapi.sdos	MAPI	n/a	None
sdooutlook.sdos	Outlook	olapp & olns	Outlook.Application & Outlook.Namespace objects
sdosystem.sdos	System	n/a	None
sdownword.sdos	Word	word	Word.Application object

Rules for Returning Values

In VBScript, functions can return values by setting the function name to a value. For example, the custom_archive function would return a value with the line “custom_archive = “value””.

A function can and should return one or more lines of text for any function whose name begins with “get”. Multiple lines should be separated by vbCrLf (or chr(13)+chr(10)).

If the name does not begin with “get”, returned values other than error messages are ignored.

Naming Conventions

To avoid potential conflicts with sdOffice command names that may be provided in the future, you should adopt a naming convention that adds a prefix and underscore to your function names. This can be as simple as “x_name”, or something more explicit, like “custom_name”. VBScript does not allow spaces in function names. If you wish to include a space, use an underscore ‘_’ instead.

Distribution of Script Files

A method is provided to automatically distribute script files to each client as they connect, using the server’s manifest.txt file as described in the [Automated File Distribution](#) chapter.

Support

SDSI cannot offer support for how to write VBScript functions or control Microsoft’s Office objects. There are many books on VBScript, and a great deal of documentation available from

Microsoft regarding the programming of Office applications. We recommend recording macros and using the resulting generated VB code for examples of using the objects of the various Office products.

AUTOMATED FILE DISTRIBUTION

The sdOffice Network Server provides support for file transfers to and from Office clients. On the server side, these files must reside in the “files” subdirectory under the server’s installation path. There is a special file in this directory called “manifest.txt”, which is automatically read and processed by Office Clients, either once on start up or at intervals. This file can be structure to provide automatic distribution of files to sdOffice client PC’s.

The structure of this file is similar to a .ini file, with section headers enclosed in square brackets. Each section header is composed of one or more semi-colon delimited values that represent target wild cards. These target values can be in one of three forms:

- *@machinename*, where *machinename* must match the name of the Office Client system name. This value can be a wildcard, such as @Sacto*.
- *~username*, where *username* is the login user name on the Office Client system. This can be a wildcard, such as ~Admin*
- An IP address or wildcard, such as 192.168.1.10 or 192.168.1.*, or just * to match every machine.

When clients read the manifest file, each section header is evaluated to determine if the local machine and/or user matches one of the targets. If not, the section is skipped, but if it does match, the section is processed.

To process the section, each line is read. Lines in the format *file=path* are parsed, and the *file* found in the server’s “files” directory is tested. If it is new or has changed since the last time the client read it, the file is copied to the client’s *path*. The *path* can contain special folder names in brackets. A valid line, for example, might be **announce.pdf=[desktop]announce.pdf**, to place the named file to each user’s desktop. See the [getpath](#) command for a list of valid special folder names.

An example of a manifest.txt file might look like this:

```
[*]
# files to maintain on all PCs
# serverfile=PC path
announcements.pdf=[desktop]announce.pdf

[192.168.1.*]
# all PC's in this intranet
announcements.pdf=[desktop]announce.pdf

[@sales01;@sales02;~Administrator]
# several specific machines, and an administrator user
NewClients.pdf=[mydocuments]NewClients.pdf
PartsCatalog.pdf=[mydocuments]Parts Catalog.pdf
```

One use for the manifest.txt file is to distribute common versions of custom script files. By placing copies of updated scripts into the files directory, and placing entries in the manifest.txt file targeting the [plugins] special folder, script files can be distributed automatically to all users.

For example, if there is an update to the ADO script file, you could add a section like this:

```
[*]  
SDOADO.sdos=[plugins]SDOADO.sdos
```

APPENDIX

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)