Sesame Application Programming Interface (Sesame API) Guide

A comprehensive guide to Sesame's application programming interface

LANTICA SOFTWARE

Documentation version 1.0

Acknowledgements
SesameAPI uses parts of the iMatix SFL package, Copyright © 1991-2000 iMatix Corporation
http://www.imatix.com.

LANTICA SOFTWARE LICENSE AND WARRANTY

IMPORTANT: PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE USING THE SOFTWARE. LANTICA SOFTWARE, LLC AND/OR ITS SUBSIDIARIES ("LANTICA") IS WILLING TO LICENSE THE SOFTWARE TO THE INDIVIDUAL, COMPANY OR OTHER LEGAL ENTITY THAT WILL BE USING THE SOFTWARE (REFERENCED BELOW AS "YOU" OR "YOUR") ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS OF THIS LICENSE AGREEMENT. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU AND LANTICA. BY OPENING THIS PACKAGE, BREAKING THE SEAL, OR LOADING THE SOFTWARE, YOU AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, MAKE NO FURTHER USE OF THE SOFTWARE AND REMOVE ANY INSTALLATION FROM YOUR SYSTEM.

1. License
The software which accompanies this license (collectively, the "Software") is the property of Lantica and/or its licensors and is protected by U.S. and international copyright law. While Lantica continues to own the Software, you will have certain rights to use the Software after you accept this license. This license governs any releases, revisions, or enhancements to the Software that Lantica may furnish to you. You may be held legally responsible for any copyright infringement which is caused or encouraged by your failure to abide by the terms of this license. Except as may be modified by a Lantica license certificate, license coupon, license key (each a "License Module") which accompanies, precedes, or follows this license, your rights and obligations with respect to the use of the Software are as follows:

You may not:
A. Copy the printed documentation that accompanies the Software;
B. Sublicense, sell, lease, or rent any portion of the Software except as included as part of a derived work;
C. Reverse engineer, decompile, disassemble, modify, adapt, translate, make any attempt to discover the source code of the Software; or
D. Use the Software in any manner not authorized by this license.

2. Limited Warranty:
Lantica does not warrant that the Software will meet your requirements or that the operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE AND COUNTRY TO COUNTRY.

3. Disclaimer of Damages:
SOME STATES AND COUNTRIES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE BELOW LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW AND REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL LANTICA OR ITS LICENSORS BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT, PUNITIVE, INCIDENTAL OR SIMILAR DAMAGES, INCLUDING BUT NOT LIMITED TO ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE, EVEN IF LANTICA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL LANTICA'S OR ITS LICENSORS' LIABILITY EXCEED THE PRICE THAT YOU PAID FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept this license or not.

4. General:

This license will be governed by the laws of the Commonwealth of Pennsylvania, United States of America, notwithstanding any conflicts rules to the contrary. This license agreement and any related License Module is the entire agreement between you and Lantica relating to the Software and supercedes all prior or contemporaneous oral or written communications, proposals and representations with respect to its subject matter, and shall prevail over any conflicting or additional terms of any quote, order, acknowledgement, or similar communications between the parties.
This agreement may only be modified by a License Module or by a written document which has been signed by both you and Lantica. This license will automatically terminate without notice from Lantica if you fail to comply with any of the terms contained herein. You may terminate this license at any time by giving written notice of termination to Lantica. Upon any termination of this license, you must cease all use of and destroy all copies of the Software. The disclaimers of warranties and damages and limitation on liability shall survive any termination of this license. Should you have any questions concerning this license, or if you desire to contact Lantica for any reason, please write to: Lantica Customer Service, P.O. Box 27, Penns Park, PA 18943-0027, USA

Table of Contents

Sesame Application Programming Interface Guide

About the Sesame Application Programming Interface (Sesame API)

The Sesame Application Programming Interface (Sesame API) is a set of functions and subroutines used to access an application created with Sesame Database Manager and running on an active Sesame Server.

The procedures provided by the Sesame API duplicate a selection of the functions provided by SBasic - the programming language built into Sesame Database Manager - including the XResultSet family of commands and the @Error command.

For complete information about Sesame Database Manager, see the Sesame User Guide supplied with Sesame Database Manager. For complete information about the SBasic commands exposed by this API, see the Sesame Programming Guide supplied with Sesame Database Manager.
Note: All libraries supplied with Sesame API are 32 bit.

The Sesame API is provided at no charge as an additional tool for use with Sesame Database Manager. Due to the wide variation in system configurations and language versions, we cannot provide free technical support for installation or use of the Sesame API. If you need assistance with the Sesame API, please contact Lantica Software Technical Support to schedule a paid Focused Support session or paid consulting assistance from the API Development Team.

Installing the Sesame API

The Sesame API package is provided as a self-extracting exe file for Microsoft Windows and as a gzipped tar file for Linux. These can be extracted wherever you wish and the various examples and libraries copied into place as needed. To match the paths used in the example files - which use the default install location for Sesame - extract to the following locations:

Windows: `C:\Sesame2\Utilities\Lantica`
Linux: `/usr/local/Sesame2/Utilities/Lantica`

A SesameAPI folder will be created in the extract location with a `docs` subfolder containing documentation, a `sample_data` folder with Sesame sample applications, and a subfolder for each supported language. C/C++ and Visual Basic users should use the libraries in the subfolder called shared. Each language subfolder contains the following for that language:

```
example files
include files and declarations
.dll, .lib and .so files
version.txt file with information about library and language versions
```

Starting a Sesame Server

Important: The Sesame Server must be version 2.5 or higher. Earlier versions of Sesame cannot accept a Sesame API connection.

The Sesame API is a type of Sesame Client. Just like a standard Sesame Client, it must connect to a running Sesame Server to function and any Sesame database applications you want to use must be accessible to that Sesame Server. You must also have enough client licenses to allow the Sesame API Client to connect.

You can connect to a Sesame Server you start manually, or you can make use of system or shell commands to start a Sesame Server as part of your implementation,

as appropriate. For complete instructions on starting a Sesame Server, see the Sesame User Guide supplied with Sesame Database Manager.

Sesame API Commands Quick Reference

Following is a Quick Reference that lists all the built-in commands available for use in Sesame API programming.

SesameClose(rs)	Closes a result set.
SesameConnect(server)	Connect to a Sesame Server
SesameCreateNewRecord(rs)	Creates a new record in a result set.
SesameDeleteRecord(rs)	Deletes a record in a result set.
SesameDisconnect()	Disconnect from the currently connected Sesame Server.
SesameError()	Returns information about the most recent run time error.
SesameGetCurrentPosition(rs)	Returns the current record position for a result set.
SesameGetValue(rs, fn)	Returns a field value from a record in a result set.
SesameLocked(rs)	Checks whether the current record in the specified result set is locked.
SesameParent(rs)	Returns a handle to a result set containing a single record which is the natural parent of the current record in result set rs.
SesameRemoveRecord(rs)	Removes a record from a result set. This does not delete the record from the database.
SesameReparent(rs_parent, field_name, rs_child)	Sets the records in a result set to be children of the current record in another result set.
SesameRunProgram(rs, global_program, program, test_only)	Runs a SBasic program on every record in a result set. Similar to running a Mass Update.
SesameSearch(filename, db, search_mode, search_syntax, criteria, username, password)	Creates a result set based on retrieve criteria. The value returned is used as the result set handle required by the Sesame API commands.
SesameSetCurrentPosition(rs, pos)	Sets the current record position for a result set. Can be used in a loop to iterate through the records in the result set.
SesameSetValue(rs, fn, val)	Sets the value of a field in a record in a result set.
SesameSort(rs, sort_vals)	Sorts a result set.
SesameSubSet(rs, fn)	Returns a handle to a result set containing the natural subrecords of the current record in result set rs.
SesameTotal(rs)	Returns the number of records in a result set.
Special Commands	
SesameAltGetValue(rs, fn, limit, output)	Alternate version of SesameGetValue() for use with Visual Basic.
SesameAltRunProgram(rs, global_program, program, test_only, limit, output)	Alternate version of SesameRunProgram() for use with Visual Basic.

Alphabetic Command Reference

This section provides detailed reference information and usage examples of these built-in commands. Each command lists its syntax, parameters, return value type, and corresponding SBasic function. Most commands also include a detailed description of what the command does. Further information about a command may be available in the documentation for the corresponding SBasic function in the Sesame Programming Guide supplied with Sesame Database Manager.

Note: The Sesame API is available in a number of languages. Code examples shown are illustrative and may not match the specific syntax of your preferred language.

SesameClose(rs)

Parameters: rs as 32 bit integer
Returns: Nothing
SBasic Function: XResultSetClose()

SesameClose closes a result set, signaling to Sesame that you are done using that result set. If you open a result set by any means, you must call SesameClose on that result set.

This command takes as its only argument the "handle" to the result set.

```
rs == SesameSearch("Customers.db", "Customers", 0, 2, "", "", "", "");
if(rs > -1)
{
    // Do stuff
    SesameClose(rs);
}
```

SesameConnect(server)

Parameters: server as string
Returns: 32 bit integer
SBasic Function: None. Corresponds to opening a standard Sesame client.

SesameConnect opens a connection to a running Sesame server. The `server` argument is the name (and, optionally, port numbers) of the Sesame Server to which to connect. A return value greater than zero (0) indicates a successful connection.

For more information on Sesame connection syntax, see the Network Administration section of the Sesame User Guide supplied with Sesame Database Manager.

```
flag == SesameConnect(my_server);

flag == SesameConnect(myserver:20000:20001);
```

Note: If SesameConnect() is successful, you must call SesameDisconnect() when you are finished to release the connection and free up the client license.

SesameCreateNewRecord(rs)

Parameters: rs as 32 bit integer
Returns: Nothing
SBasic Function: XResultSetCreateNewRecord()

SesameCreateNewRecord creates a new record in the result set and sets the current record to the newly created one. After it is created, fields can be filled out using SesameSetValue. The values you set in the new record do not have to match the search spec that created the result set.

This command takes as its only argument the "handle" to the result set.

```
rs == SesameSearch("Customers.db", "Customers", 0, 2, "", "", "",
"!Key=<0");
if(rs > -1)
{
    SesameCreateNewRecord(rs);
    SesameSetValue(rs, "Key", "1000");
    SesameSetValue(rs, "City", "Albany");
    SesameClose(rs);
}
```

SesameDeleteRecord(rs)
Parameters: rs as 32 bit integer
Returns: Nothing
SBasic Function: XResultSetDeleteRecord()

SesameDeleteRecord permanently deletes the record from the database.

This command takes as its only argument the "handle" to the result set.

Be careful using this command inside a loop using SesameSetCurrentPosition and SesameTotal. Both values will be changed because a record has been deleted from the result set. When a record is deleted, the current record will become the one following the deleted record, if there is one. If there are no records following the deleted record, the current record will fall back to the previous record.

```
// Deletes all records where State = "MA"
rs == SesameSearch("Customers.db", "Customers", 0, 2, "", "", "",
"State=MA");
if(rs > -1)
{
    while(SesameTotal(rs) > 0)
    {
            SesameDeleteRecord(rs);
    }
    SesameClose(rs);
}
```

SesameDisconnect()
Parameters: None
Returns: Nothing
SBasic Function: None. Corresponds to closing a standard Sesame client.

Closes the currently active connection created with SesameConnect(). This releases the connection and frees up the client license used by the connection.

```
flag == SesameConnect(my_server);
if(flag > 0)
{
    // Do stuff
    SesameDisconnect();
}
```

SesameError()
Parameters: None
Returns: 32 bit integer
SBasic Function: @ErrorType

Returns a code describing the latest run time error. A return value other than zero (0) indicates that an error has occurred. Errors are defined as follows:

```
NO_ERROR                    0
NO_FORM_FOUND               1
```

```
NULL_WIDGET                     2
OUT_OF_RANGE_WIDGET_INDEX       3
OUT_OF_RANGE_FIELD_NUMBER       4
NO_PERMISSION                   5
NULL_FORM                       6
NULL_REPORT                     7
NULL_LAYOUT                     8
NULL_VALUE                      9
FORM_REQUIRED                   10
INAPPROPRIATE_LE                11
ILLEGAL_COLOR                   12
NO_REPLY                        13
COMMAND_FAILURE                 14
PREVIEW_MODE                    15
MEMORY_ALLOCATION_ERROR         16
NULL_SELECTION                  17
CANNOT_OPEN_FILE                18
CANNOT_ACCESS_FILE              19
NULL_APPLICATION_FILENAME       20
ILLEGAL_VALUE                   21
NULL_OPTION                     22
FIELD_EDITOR_UP                 23
MACRO_STACK_OVERFLOW            24
NULL_MACRO_STACK                25
NULL_MACRO_FILENAME             26
MASS_UPDATE                     27
MACRO_STACK_UNDERFLOW           28
NULL_COMMAND_TREE               29
NULL_LABEL                      30
LAYOUT_REQUIRED                 31
NO_FORM_WIDGET                  32
NO_WIDGET_COUNT                 33
REQUIRED_UI_ELEMENT             34
RECORD_CREATION_FAILURE         35
RECORD_OUT_OF_RANGE             36
GENERAL                         37
IMAGE_FILE_ERROR                38
PRINTER_DC                      39
UNSUPPORTED                     40
REPORT_REQUIRED                 41
PARAMETER_ERROR                 42
NULL_RESULT_SET_KEY             43
ILLEGAL_RESULT_SET_KEY          44
REENTRANT                       45
OUT_OF_RANGE                    46
UNEXPECTED_NULL                 47
ILLEGAL_DB_ID                   48
FILE_EXISTS                     49
ELEMENT_NOT_FOUND               50
DIRECTORY_DOES_NOT_EXIST        51
FAILED_TO_CREATE_DIRECTORY      52
INCORRECT_FORM_MODE             53
REGEX_SYNTAX_ERROR              54
RUNTIME_COMPILATION_ERROR       55
RUNTIME_EXECUTION_ERROR         56
FTP_PROTOCOL_ERROR              57

rs == SesameSearch("Customers.db", "Customers", 0, 2, "", "", "",
"!Key=<0");
if(rs > -1)
{
    SesameCreateNewRecord(rs);
    if(SesameError() == 0)
    {
           SesameSetValue(rs, "Key", "1000");
           SesameSetValue(rs, "City", "Albany");
    }
    else
    {
           // Show error message "Failed to create new record."
    }
    SesameClose(rs);
}
```

SesameGetCurrentPosition(rs)
Parameters: rs as 32 bit integer
Returns: 32 bit integer
SBasic Function: @XResultSetCurrentPosition()

SesameGetCurrentPosition returns the result set position of the "current" record as set by the command SesameSetCurrentPosition. Records are numbered from 1 to the total number of records in the result set, which can be determined using SesameTotal.

SesameGetCurrentPosition takes as its only argument the "handle" to the result set.

```
pos == SesameGetCurrentPosition(rs);
```

SesameGetValue(rs, fn)
Parameters: rs as 32 bit integer, fn as string
Returns: string
SBasic Function: @XResultSetValue()

This function returns the field value of the specified field as a string. It obtains this field from the current record.

This command takes as its first argument the "handle" to the result set. The second argument is the field name of the field to return.

Note to Visual Basic users: This function will not work in Visual Basic as Visual Basic cannot receive strings as return values from certain types of DLLs. Use SesameAltGetValue() instead.

```
val == SesameGetValue(rs, "My_Field");
```

SesameLocked(rs)
Parameters: rs as 32 bit integer
Returns: 32 bit integer
SBasic Function: @XResultSetLocked()

This command checks whether the current record in the specified result set is locked by another Sesame Client. It returns 0 if the record is not locked. It returns 1 if the record is locked. You should check this value before attempting to perform any write operations on the record. If the record is locked, write operations will fail.

This command takes as its only argument the "handle" to the result set.

```
if(SesameLocked(rs) == 0)
{
    SesameSetValue(rs, "City", "Albany");
}
```

SesameParent(rs)
Parameters: rs as 32 bit integer
Returns: 32 bit integer

This function is used to obtain a result set containing a single record - the natural parent of the current record in the result set passed in as the only parameter. This allows you to access the natural parent of a child record directly, even if there is no matching key value. After any operations have been completed on that single record, the parent result set should be closed. A return value greater than -1 indicates success.

```
rs = SesameSearch("Countries.db", "Cities", 0, 2, "", "", "",
"!City=Aberdeen");
if(rs > -1)
```

```
{
    parent_rs == SesameParent(rs);
    if(parent_rs > -1)
    {
           // Do stuff with the Parent record (Country: United Kingdom)
           SesameClose(parent_rs);
    }
    SesameClose(rs);
}
```

SesameRemoveRecord(rs)
Parameters: rs as 32 bit integer
Returns: Nothing
SBasic Function: XResultSetRemoveRecord()

Removes the current record from the result set, but does not delete the record from the database.

This command takes as its only argument the "handle" to the result set.

Be careful using this command inside a loop using SesameSetCurrentPosition and SesameTotal. Both values will be changed because a record has been removed from the result set. When a record is removed, the current record will become the one following the removed record, if there is one. If there are no records following the removed record, the current record will fall back to the previous record.

```
if(SesameGetValue(rs, "State") == "MA")
{
    SesameRemoveRecord(rs);
}
```

SesameReparent(rs_parent, field_name, rs_child)
Parameters: rs_parent as 32 bit integer, field_name as string, rs_child as 32 bit integer
Returns: Nothing
SBasic Function: XResultSetReparent()

SesameReparent takes the entire child result set specified by rs_child, and makes the records in that result set children of the current record in the specified parent result set. They become children linked by the specified subrecord field.

Note: This is only necessary for naturally linked children. To reparent relational children, simply change their matching field value.

This command takes three arguments. The first argument is the "handle" for a result set containing the parent record. The second argument is the field name of the subrecord field in the parent record that links to the correct child records. The third argument is a "handle" to a result set containing the child records you want to reparent.

The field name argument is the field name of the subrecord link. It is critical that you use the correct field name. This is not the element name. You can find the correct name by opening the parent form in SDesigner, right-clicking on the Subform element and choosing Subform Settings. The field name appears under Step 4 labeled Subrecord Field Name.

Because this command will reparent all of the records in the child result set, it is very important that the SesameSearch that "opens" the child result set be very precise and only include the records that you seek to reparent. You can also examine the data values in the retrieved records, and use the SesameRemoveRecord command to limit the result set to only the records you want to reparent.

As this command affects the parenting of groups of records, it is advisable to make a backup prior to using it.

```
child_rs = SesameSearch("Countries.db", "Cities", 0, 2, "", "", "",
"!City=A..");
if(rs > -1)
{
    parent_rs == SesameSearch("Countries.db", "Countries", 0, 2, "", "",
"", "!Country=United Kingdom");
    if(parent_rs > -1)
    {
           if(SesameTotal(parent_rs) == 1)
           {
                  SesameReparent(parent_rs, "Cities Subform", child_rs);
           }
           SesameClose(parent_rs);
    }
    SesameClose(rs);
}
```

SesameRunProgram(rs, global_program, program, test_only)
Parameters: rs as 32 bit integer, global_program as string, program as string, test_only as 32 bit integer
Returns: string
SBasic Function: @XResultRunProgram()

This command runs an SBasic program, specified as a string argument, on every record in a result set. It returns as a string any content written using Write or WriteLn. The programming is sent to the Sesame engine to be executed. Because it runs on the engine, the programming must use field names instead of element names. No commands that reference the user interface, forms, or reports are legal in the supplied program.

Note to Visual Basic users: This function will not work in Visual Basic as Visual Basic cannot receive strings as return values from certain types of DLLs. Use SesameAltRunProgram() instead.

For security, the File I/O commands, Shell commands and Process commands are disabled by default. These can be optionally allowed using the SERVER CODE FILE I/O and SERVER CODE SHELL INI file entries, but you should consider carefully before doing so.

NOTE: Because this method of working with a group of records works directly on the engine, it is much faster than a normal mass update, but it also is able to provide less feedback. You should always test this command using a backup of your data to make sure that is doing what you intend.

Arguments:
rs - "Handle" to a result set

global_program - Programming that you would type into the GLOBAL CODE area. Must be written in SBasic.

program - Programming to run on each record in the result set. Must be written in SBasic.

test_only - Flag indicating whether to actually run the program. 0 runs the program. 1 tests whether the program compiles without running it.

As the program compiles on the engine, the syntax error interface available in the Sesame Programming Editor is not available, however, SesameError will be set if the program fails to compile. To test your program, set test_only to 1 and check SesameError.

```
str = SesameRunProgram(rs, "", pgm, 0)
```

SesameSearch(filename, db, search_mode, search_syntax, criteria, username, password)
Parameters: filename as string, db as string, search_mode as 32 bit integer, search_syntax as 32 bit integer, criteria as string, username as string, password as string
Returns: 32 bit integer
SBasic Function: @XResultSetSearch()

This command is the primary way to obtain a new result set on which the other Sesame API commands will operate. This function performs a search in the specified database and, upon completion of the search, returns a "handle" to the result set containing the records that match the search. A return value greater than -1 indicates success.

SesameSearch takes seven arguments:

filename - the path and filename of the application to open.

db - the name of the database in that application in which to search.

search_mode - specifies whether records must match all or any of your criteria to be included. Valid values are as follows:
0: Records must match all of the criteria
1: Records must match any of the criteria

search_syntax - specifies whether the criteria use Q&A compatible syntax or Regular expressions syntax. Full documentation of Q&A search syntax is available in the Sesame User Guide supplied with Sesame Database Manager. Valid values are as follows:
2: Q&A compatible syntax
3: Regular expressions syntax

criteria - one or more parameters, each having the form "!fieldname=spec". These specs provide the criteria that determine which records are included in the result set. The "spec" portion is whatever you would have typed into the Sesame retrieve spec. As this command accesses the underlying database directly, use database names and field names.

Separate multiple criteria with %. Do not put extra quote marks around specs with spaces. Do not put spaces around the equal sign between the field name and the spec or between the bang (!) and the field name.

Examples (Q&A syntax):

```
!Key=1000                          // Finds records where Key equals 1000
!Key==                             // Finds records where Key is blank
!Key=/=                            // Finds records where Key is not blank
!City=San Francisco%!Key=>1000     // Finds records where City
                                   // equals "San Francisco" and Key
                                   // is greater than 1000
```

To create a result set including all records in the database, use a zero-length string ("") as the criteria.

To create an empty result set, use criteria that will match no records, such as searching for records where a field that should never be blank is blank.

If you specified Q&A syntax in the search_syntax parameter, use Q&A syntax in your specs. If you specified regular expressions syntax in the search_syntax parameter, use regular expressions syntax in your specs.

username - username allowing access to the desired application and database. If the application does not use security, pass a zero-length string ("").

password - password allowing access to the desired application and database for the specified username. If the application does not use security, pass a zero-length string ("").

Usage Examples:

```
rs == SesameSearch("Customers.db", "Customers", 0, 2, "", "", "", "")

rs == SesameSearch("Data\Customers.db", "Customers", 0, 2, "", "!City=San
Francisco%!Key=>1000", "", "")

rs == SesameSearch("Data\Customers.db", "Customers", 0, 2, "",
"!Key=>1000", "my_username", "my_password")
```

In the examples above, after the search is performed, a "handle" to the result set is returned to RS. RS is then used to operate on the result set. **It is very important that you close any result set you open using the SesameSearch command after you are finished with it by calling SesameClose() on that result set.**

Of the two types of retrieve spec programs usable in Sesame retrieve specs, this command supports one. You can use a retrieve spec program as long as the program does not reference any fields. So, for example you can check if a date is younger than 1000 days old using:

```
rs == SesameSearch("Customers.db", "Customers", 0, 2,
"!Date_Entered=>{@Date - 1000}")
```

But you cannot check a different field using:

```
rs == SesameSearch("Customers.db", "Customers", 0, 2,
"!Company={Date_Entered > (@Date - 1000)}")
```

If using a retrieve spec program, the program must be written in SBasic.

SesameSetCurrentPosition(rs, pos)
Parameters: rs as 32 bit integer, pos as 32 bit integer
Returns: Nothing
SBasic Function: XResultSetCurrentPosition()

SesameSetCurrentPosition tells Sesame which record in the result set should be current. It is the current record that is affected by the other Sesame API commands. Typically, you would use this command in a loop from 1 to the value returned by SesameTotal to operate on all of the records in the result set.

This command takes as its first argument the "handle" to the result set. The second argument is the position of the record you want to make current.

```
SesameSetCurrentPosition(rs, 5)
```

SesameSetValue(rs, fn, val)
Parameters: rs as 32 bit integer, fn as string, val as string
Returns: Nothing
SBasic Function: XResultSetValue()

This command sets the value of the specified field in the current record to the specified value.

This command takes as its first argument the "handle" to the result set. The next argument is the field name of the field to set. The last argument is the actual value to set.

This example sets the FullName field.

```
SesameSetValue(rs, "Company", "ABC Corporation")
```

Note: Since the values are placed in the database fields directly, they do not go through the formatting and type checking as they do when you type them into a form. If you set a field value that is not the correct type, the results will be unpredictable. , For example, if you post a non-date value to a date field, it will actually keep that value even if it cannot be determined to be a date.

```
SesameSetValue(rs, "Date_Entered", "ABC Corporation")
```

The example above will put "ABC Corporation" (an invalid date) in the Date_Entered field. In form view in Sesame, such a date would usually appear blank.

SesameSort(rs, sort_vals)
Parameters: rs as 32 bit integer, sort_vals as string
Returns: Nothing
SBasic Function: XResultSetSort()

Sorts a result set.

This command takes as arguments the "handle" to the result set and the list of sort values.

The sort_vals argument is a semicolon-separated list with the form field_name:direction.

field_name - The name of the underlying database field by which to sort. Note that this is the field name, not a layout element name.

direction - The sort direction. Use -1 for ascending sort. Use 1 for descending sort. For example, "LastName:-1;FirstName:1" - sorts first by LastName in ascending order and then by FirstName in descending order.

```
SesameSort(rs, "Key:-1")
```

SesameSubSet(rs, field_name)
Parameters: rs as 32 bit integer, field_name as string
Returns: 32 bit integer
SBasic Function: @XResultSetSubset()

This function accepts a currently open result set handle and the name of a subrecord field. It returns a handle to a result set representing the naturally linked records that are subrecords of the current parent record in the open result set. A return value greater than -1 indicates success.

This function accepts two arguments: the handle of the parent result set and the name of the SUBRECORD field in the parent record that defines the natural link. This allows you to quickly access the natural children of a given parent record without needing to do key-based lookups.

```
// Get a parent record
```

```
parent_rs == SesameSearch("Countries.db", "Countries", 0, 2,
"!Country=United Kingdom")
if(parent_rs > -1)
{
    // Get child records
    child_rs == SesameSubSet(parent_rs, "Cities Subform")
    if(child_rs > -1)
    {
           // Get the count of cities for this country
           count == SesameTotal(child_rs)
           SesameClose(child_rs)
    }
    SesameClose(parent_rs)
}
```

SesameTotal(rs)
Parameters: rs as 32 bit integer
Returns: 32 bit integer
SBasic Function: @XResultSetTotal()

This function returns the total number of records in the result set. Its only argument is the "handle" to the result set.

```
count == SesameTotal(child_rs)
```

Special Command Reference

SesameAltGetValue(rs, fn, limit, output)
Parameters: rs as 32 bit integer, fn as string, limit as 32 bit integer, output as string
Returns: 32 bit integer
SBasic Function: @XResultSetValue()

Note: This function is a special version of SesameGetValue() specifically for languages like Visual Basic which cannot receive strings as return values from certain types of DLLs. If you are not using one of these languages, use SesameGetValue() instead.

This function returns the length of the value in the specified field. It obtains this field value from the current record. When this function runs, the output parameter is set to the field value itself.

This command takes as its first argument the "handle" to the result set. The second argument is the field name of the field to return. The third argument, `limit`, is the maximum length of value to write to output. The `output` argument is a string which can accept a value of a length no less than limit.

To use this function in Visual Basic, declare a string of the same length as `limit`. After you run SesameAltGetValue, use the return value to close up the empty space at the end of `output`.

```
Dim strOut As String * 1024
Dim lngLen As Long
Dim strVal as String


    ' Get value from the Sesame record
    lngLen = SesameAltGetValue(lngRS, "MyField", 1024, strOut)

    ' Strip off the extra space
    strVal = Left(strOut, lngLen)
```

SesameAltRunProgram(rs, global_program, program, test_only, limit, output)

Parameters: rs as 32 bit integer, global_program as string, program as string, test_only as 32 bit integer, limit as 32 bit integer, output as string
Returns: 32 bit integer
SBasic Function: @XResultRunProgram()

Note: This function is a special version of SesameRunProgram() specifically for languages like Visual Basic which cannot receive strings as return values from certain types of DLLs. If you are not using one of these languages, use SesameRunProgram() instead.

This command runs a SBasic program, specified as a string argument, on every record in a result set. It returns as a string any content written using Write or WriteLn. The programming is sent to the Sesame engine to be executed. Because it runs on the engine, the programming must use field names instead of element names. No commands that reference the user interface, forms, or reports are legal in the supplied program.

For security, the File I/O commands, Shell commands and Process commands are disabled by default. These can be optionally allowed using the SERVER CODE FILE I/O and SERVER CODE SHELL INI file entries, but you should consider carefully before doing so.

NOTE: Because this method of working with a group of records works directly on the engine, it is much faster than a normal mass update, but it also is able to provide less feedback. You should always test this command using a backup of your data to make sure that is doing what you intend.

Arguments:
rs - "Handle" to a result set

global_program - Programming that you would type into the GLOBAL CODE area. Must be written in SBasic.

program - Programming to run on each record in the result set. Must be written in SBasic.

test_only - Flag indicating whether to actually run the program. 0 runs the program. 1 tests whether the program compiles without running it.

limit - the maximum length of value to write to output

output - a string which can accept a value of a length no less than limit

As the program compiles on the engine, the syntax error interface available in the Sesame Programming Editor is not available, however, SesameError will be set if the program fails to compile. To test your program, set test_only to 1 and check SesameError.

```
Dim strOut As String * 1024
Dim lngLen As Long
Dim strVal as String


    ' Run program and receive return value
    lngLen = SesameRunProgram(rs, "", pgm, 0, 1024, strOut)

    ' Strip off the extra space
    strVal = Left(strOut, lngLen)
```

Specific Language Notes and Sample Files

Language Version Issues

In each language case, the Sesame API is compiled against a common version of the library for that language. Some languages have a higher tolerance than others for version mismatches. If you attempt to use the Sesame API and encounter a version mismatch issue, please contact Lantica Software Technical Support to request a version compiled for your system.

Visual Basic/Visual Basic for Applications

Use the samples and libraries in the `shared` folder. Examples include a Microsoft Excel spreadsheet and two Microsoft Word merge documents. To see the code behind any of these samples, open the Word or Excel file and press `Alt-F11` to open the Visual Basic Code Editor.

Samples:
api.xls - Excel Spreadsheet. Contains two macros. Press `Alt-F8` to select and run a macro. Before doing so, edit the marked settings in the code to match your system.
UpdateWorksheetFromSesame - Gets values from Sesame records and fills columns in the spreadsheet.
UpdateSesameCreditLimit - Picks up values from a column in the spreadsheet and writes to the Sesame database.

CreditLimitMerge.doc - Word merge document. Macro will run as soon as the document is opened. An initial confirmation allows you to prevent the merge until you have edited the settings to match your system. Uses the API to pull Sesame data and write the merge data file without needing to export from Sesame first. Merges to new document.

CountriesMerge.doc - Word merge document. Macro will run as soon as the document is opened. An initial confirmation allows you to prevent the merge until you have edited the settings to match your system. Uses the API to pull Sesame data and write the merge data file without needing to export from Sesame first. Demonstrates using subrecords. Merges to new document.

To use the Sesame API in your own VB or VBA, you must declare the dll procedures. A text file called `declarations.txt` is located in the `shared` folder. You can copy the declarations for any Sesame API procedure you intend to use from there.

Visual Basic cannot accept strings as return values from certain kinds of dlls, including the Sesame API. Special alternate functions have been provided for use with Visual Basic. These are listed in the *Special Commands* section of this document.

C/C++

To use Sesame API procedures in your code, include `shared\sesame_c_shared.h`. When you compile for Windows, include `shared\sesame_api.lib` on your compile line or in your compile environment. To run your program, `shared\sesame_shared_api.dll` (Windows) or `shared/libsesame_shared_api.so` (Linux) needs to be copied to your working directory or a directory in your library path.

Samples:
sesame_api_c_example.cpp - Connects to the Sesame Server and sends data to standard output based on key values supplied from standard input. A file with sample data - `records.txt` - is included. Before compiling, change the marked settings to match your system. If compiling for Linux, define UNIX (see example below).

Usage Examples:
```
> cat records.txt | sesame_api_c_example
> sesame_api_c_example < records.txt
> type records.txt | sesame_api_c_example
```

Example compile command lines:

```
> cl sesame_api_c_example.cpp sesame_api.lib

> g++ -DUNIX sesame_api_c_example.cpp -L. -lsesame_shared_api -o
sesame_api_c_example
```

PHP

To use Sesame API procedures in your code, include `php\sesame_php_api.php`. To run your program, `php\sesame_php_api.dll` (Windows) or `php/sesame_php_api.so` (Linux) needs to be copied to your php extension directory and the extension loaded. A sample `php.ini` file is included in the `php` folder.

Samples:
main.php - Connects to Sesame Server, retrieves information from the sample Customers application, and prints values from the retrieved records.

sample_site - Subdirectory containing a set of .php and .html files demonstrating how to search, add and edit Sesame records in the sample Customers application using the Sesame API behind a web interface.

Perl

To use Sesame API procedures in your code, load `perl\sesame_perl_api.pm`. To run your program, `perl\sesame_perl_api.dll` (Windows) or `perl/sesame_perl_api.so` (Linux) needs to be copied to an accessible directory.

Samples:
main.pl - Connects to Sesame Server, retrieves information from the sample Customers application, and prints values from the retrieved records.

Python

To use Sesame API procedures in your code, import `python\sesame_python_api.py`. To run your program, `python\_sesame_python_api.pyd` (Windows) or `python/_sesame_python_api.so` (Linux) needs to be copied to an accessible directory.

Samples:
main.py - Connects to Sesame Server, retrieves information from the sample Customers application, and prints values from the retrieved records.

Java

Before use, you must compile `java\sesame_java_api.java` and `java\sesame_java_apiJNI.java`. To use Sesame API procedures in your code, load `sesame_java_api`. To run your program, `java\sesame_java_api.dll` (Windows) or `java/libsesame_java_api.so` (Linux) needs to be copied to an accessible directory.

Samples:
main.java - Connects to Sesame Server, retrieves information from the sample Customers application, and prints values from the retrieved records.

C#

To use Sesame API procedures in your code, compile in `c_sharp\sesame_csharp_api.cs` and `c_sharp\sesame_csharp_apiPINVOKE.cs`. To run your program, `c_sharp\sesame_csharp_api.dll` (Windows) or `c_sharp/libsesame_csharp_api.so` (Linux) needs to be copied to your working directory or a directory in your library path.

Samples:
main.cs - Connects to Sesame Server, retrieves information from the sample Customers application, and prints values from the retrieved records.

Example compile command lines:

```
> csc main.cs sesame_csharp_api.cs sesame_csharp_apiPINVOKE.cs
```

Reader's Notes

Reader's Notes