

STN Express[®]

Version 8.3 for Windows[®]

User Guide

March 2008



CAS is a division of the
American Chemical Society

Copyright © 2008 American Chemical Society
All rights reserved.

Appendix B - STN Express Script Language

Steps in Writing and Running a Script	B-1
Defining the Task	B-1
Translating the Task into Script Statements	B-2
Creating a Script	B-2
Saving the Script	B-3
Checking the Script for Errors	B-3
Using or Running the Script	B-3
Script Language Components	B-4
Variables	B-4
Character Strings	B-5
Numbers	B-6
File Names	B-6
Labels	B-6
Operators and Conditions	B-7
Arithmetic Operators	B-7
String Operators	B-7
Conditional Operators	B-7
Compound Conditions	B-8
Special Operators	B-8
Syntax Requirements	B-9
Statements	B-10
How Statements are Described	B-10
=> :	B-11
Assignment Statement	B-12
BEGIN/END	B-12
BREAK	B-12
CAPTURE	B-13
CLOSE	B-13
CONTINUE	B-14
DELETE	B-14
DROP DTR	B-14
ECHO	B-15
EDIT	B-16
EXEC	B-16
EXIT	B-17
GET	B-18
GOSUB	B-19
GOTO	B-20

IF	B-20
ONEXIT	B-21
OPEN	B-22
PAGE	B-22
PAUSE	B-23
PRINT	B-23
RAISE DTR	B-23
READ	B-23
RETURN	B-24
SEND	B-24
STNLOGON	B-25
SUBSTR	B-25
TRYAGAIN	B-26
UPLOAD	B-26
USER	B-27
VT320	B-27
WAIT	B-27
WRITE	B-28
Sources of Example Scripts	B-28
Script Tips	B-28
Registry Number Script Example	B-30
Author Search Script Example	B-31
Logon Scripts	B-33
Predefined Logon Setup Variables	B-33
Setup Definition Dialog	B-33
Advanced Dialog	B-34
Predefined STN Variables	B-34

Appendix C - MDL Molfile Conversion Assumptions

Appendix D - Access to the Merged Markush Service from Questel*Orbit

B

Appendix: STN Express Script Language

A script is a text file that contains a set of statements in the Script Language that are executed by STN Express with *Discover!* The STN Express with *Discover!* Script Language can be used to create:

- custom logon procedures
- custom search strategies
- procedures to automate repetitive search or display tasks.

For example, you may routinely perform a certain subject or author search on STN. Rather than logging on to STN and opening a transcript file to capture search results, searching multiple files, and then logging off and reviewing the transcript, you can use scripts to streamline the process. One script can log you on to STN, another can initiate the searches, and another script can display your answers. Alternatively, one script can perform all three tasks. Because the script runs automatically, without waiting for you to type, you may reduce your online time and costs.

Steps in Writing and Running a Script

Defining the Task

Define what you want to accomplish and the steps required to accomplish the task. For example, suppose you have a list of CAS Registry Numbers (RN) and want to display their records on STN. What steps would you follow?

1. Log on to STN.
2. Enter the Registry file.
3. Display the record for the first RN.
4. Repeat step 3 until finished.
5. Log off STN.

Translating the Task into Script Statements

After you have defined the task, translate the steps into Script Language statements that will accomplish the task. (The statements are listed in detail later in this document.)

1. Log on to STN.
The script could log on to STN, but it is often better to separate the logon task from the job the script performs. In this example, log on to STN as usual and then run the script.

2. Enter the Registry file.
The STN command to enter the Registry file is FILE REGISTRY. To use an STN command in a script, use the => (arrow prompt) statement:

```
=> FILE REGISTRY
```

3. Display the record for the first Registry Number.
To display a record on STN, use the DISPLAY command. Again we use the => statement:

```
=> DIS 50-00-0
```

4. Repeat step 3 until finished.
Add more DIS commands.

5. Log off STN.
End your STN session with the LOGOFF command, again using the => statement:

```
=> LOG Y
```

Creating a Script

To create a script, select **Prepare Command File** from the **Query** menu on the STN Express Main Menu. Click **New** in the Open Script dialog. This opens the STNEDIT window with an **Untitled** text window inside it. Type the following script statements:

```
\* Display CAS RNs in the Registry file on STN.  
=> FILE REGISTRY  
=> DIS 50-00-0  
=> DIS 7777-77-7  
=> LOG Y
```

Saving the Script

To save the script file, select **Save** from the **File** menu. Type a name in the **File name** box and click **Save**. By default, the script file is saved to the User Scripts folder defined in General Preferences in the **Setup** menu (on the Main Menu Toolbar). Refer to *Appendix A* for information about General Preferences.

Do not use any name that exists in the Predefined Scripts folder unless you are replacing the functions of that script.

Checking the Script for Errors

To check the script for syntax errors, go to the **Utilities** menu in STNEdit and select **Check Command File**. If errors are found, STNEdit highlights appropriate script lines and displays a message at the bottom of the STNEDIT window. To learn the nature of the error, move the mouse pointer over the highlighted part of a line. Correct the error, save the script, and check the script again until no errors are found.

Check Command File does not find all errors because some errors cannot be found until the script actually runs. It is not necessary to be logged on to an online host to use **Check Command File**.

Using or Running the Script

While you are logged on to STN, Select **Run Command File** from the **Query** menu. A Command File dialog box is displayed. Select a script file to run and then click **Open**. The script runs until it finishes or until an error is found in the script.

You may use your own word processor to write a script as long as you save the file in a text-only format.

Statements may be entered in upper- or lowercase.

A line in a script is limited to 80 characters. However, STN commands can be up to 256 characters in length. To continue a statement on the next line, put a backslash, \, at the end of each line that is continued on the next line. A single script statement can be 140 characters long.

Scripts do not require an exact format style as you type in the statements. However, each statement **MUST** be entered on a new line. It is recommended that you use the style that is shown throughout this appendix.

Follow the steps below to begin writing a script. More advanced script examples are included after this section.

To ensure that all STN Express features are available while you are online, use the STNLOGON script instead of writing your own commands to logon to STN.

Open a new script file by selecting **New** from the **File** menu within the STNEDIT window. A new, Untitled window is displayed.

Script Language Components

The STN Express with *Discover!* Script Language is a miniature computer programming language. As such, it has many of the same components as other programming languages. The components of the Script Language include:

- **Variables** - Temporary storage places for data, or places to store data that can change
- **Character Strings and Numbers** – Constant, literal text and numeric data
- **File Names** - Names of transcript or data files on your PC
- **Labels** - Identifiers for statements
- **Statements** - Instructions to STN Express with *Discover!* that perform an action. For example, the SEND statement instructs STN Express with *Discover!* to send a string of characters to an online host or communications device.
- **Operators and Conditions** – Elements that manipulate and compare pieces of data
- **Syntax Requirements** - Rules for organizing scripts and typing statements

Variables

A variable is a name for a place to store information. Specifically, a variable can store a number, a string of characters, a file name, or the result of a search. Use a variable when a specific value is unknown or will change within a script. For example, if you write a script that performs an author search, make the script flexible by using a variable for the author's name instead of typing the author's name in the script.

Variables are named with an underscore followed by one to twelve alphanumeric characters. They are considered “declared” after their first use. Once a variable is declared, the same name may not be used to refer to another variable.

Variable names are not case-sensitive. For example, the variable named `_var` is the same as the variable named `_VAR`.

Variables are always considered to be of type string (text), except for variables that represent search results (L-numbers). Variables are automatically converted to numbers (integers) when necessary for arithmetic operations or conditions. Attempts to perform arithmetic operations on non-numeric text result in errors when the script is run.

To give a variable a value, use an “assignment statement,” described below, or the `\>` operator, described at the `=>` statement. See also the `#` operator, explained under the `=>` statement.

Up to 200 variables are permitted within a script. If there are more variables, this is noted when you run the script online rather than when you use Check Command File.

Variable values may not exceed 255 characters in length.

Character Strings

Textual data is enclosed in double quotes and can be a string up to 140 characters long.

You may continue a string on a new line by typing a backslash, \, at the end of the line.

To use the value of a string variable, type the variable along with the text. Place double quotes around the entire string.

For example:

```
_s = "how to use"
ECHO "This is _s a variable in a string."
```

displays:

This is how to use a variable in a string.

Variables within strings are replaced by their values, converting integers to text. The # operator, along with a variable, is replaced by the number of answers in the answer set and is converted to text.

If you want a variable within a string to be immediately followed by non-blank text, separate the variable name and text with a period.

For example:

```
_variable = "con"
ECHO "This example string includes
_variable.catenated text"
```

displays:

This example string includes concatenated text

Special characters and escape characters within strings include:

[CR]	Carriage return
[TAB]	Move to next tab position
[BS]	Backspace
[ESC]	Escape
[BELL]	Sound bell (beep)
[CTRL-A]...[CTRL-Z]	Unprintable control characters
[FF]	Formfeed

Note that many networks and STN Express with *Discover!* use control characters in normal operations. As a result, indiscriminate use of control characters may result in lost data, locked sessions, or terminated processes.

The characters `_`, `\`, `"`, `[`, and `#` have special use within the script language and must be doubled if they are to be used within a string. In other words, to include a `[` in a string, use two in a row:

```
ECHO "Here is a string with a left bracket\  
(i.e., [[) in it."
```

displays:

Here is a string with a left bracket (i.e., []) in it.

A distinction between upper- and lowercase is made when the characters are within a string and the number of spaces between words and letters is not ignored.

Numbers

The Script Language supports positive integers and zero; there are no decimals or fractions. Type numbers in an ordinary way. For example:

```
_var = 42  
IF _var < 100 THEN
```

To use a negative number, subtract a positive number from zero:

```
_neg = -1          \* not allowed  
_neg = 0 - 1  
IF _neg < 0 THEN
```

File Names

Enclose file names within angle brackets, `<>`. All characters within the brackets are taken literally, but underscore characters are not allowed.

File names may be stored in variables.

For example:

```
CAPTURE ON <transcript>  \* A literal file name  
_var = <new script>     \* Assign a file name  
                        \* to a variable  
EXEC _var               \* Execute a script  
                        \* whose name is in _var  
CAPTURE OFF <>
```

Labels

A "label" or "statement label" identifies a line in a script so that other statements can refer to that line. Labels are named similarly to variables except that they begin with the `@` character. The following are examples of valid labels:

```
@start  
@error
```

Refer to the `CONTINUE`, `GOTO`, and `GOSUB` statements for examples of how to use labels.

Operators and Conditions

Operators and conditions are components of statements. "Arithmetic operators" manipulate the values of numbers. "String operators" work on text strings. "Conditional operators" compare the values of strings and numbers in IF statements. "Special operators" are used in specific situations.

Arithmetic Operators

Use the arithmetic operators in assignment statements to perform arithmetic on numbers. The arithmetic operators do not function inside strings or conditions.

+	addition	add two numbers
-	subtraction	subtract one number from another
*	multiplication	multiply two numbers
/	division	divide one number by another

String Operators

+	concatenation	join two strings (not inside double quotation marks)
.	concatenation	denote the end of a variable name inside double quotes

Conditional Operators

To control execution of a statement or a set of statements based on the value of a variable, use an IF statement with a condition. A condition compares the values of two variables or a variable and a literal number or literal string.

All operators are binary, and the general syntax is A operator B.

If both operand A and operand B are integers, comparison is based on their numerical values. Otherwise, an alphabetic comparison is made.

All string comparisons are case-sensitive. That means "salt" is not equal to "SALT".

Numbers and Strings

=	equality	numbers or strings are equal
<>	inequality	numbers or strings are not equal

Numbers

>	greater than	A is greater than B
<	less than	A is less than B
>=	greater than or equal	A is greater than or equal to B
<=	less than or equal	A is less than or equal to B

Strings

INCL	includes	string A includes string B
NOTINCL	does not include	string A does not include string B

Example 1

```

_s = "string"
IF _s = "string"                \* True
IF _s = "string "              \* False
IF _s = "STRING"               \* False
IF _s INCL "ring"              \* True
IF _s INCL "Ring"              \* False
IF _s NOTINCL "cow"           \* True

```

Example 2

```

_var1 = 10
_var2 = 5 + 5
IF _var2 = _var1                \* True
IF _var2 <> var1                \* False
IF _var1 > 0                    \* True
IF _var2 < 4                    \* False
IF _var1 >= 10                  \* True
IF _var1 <= 10                  \* True

```

Compound Conditions

Compound conditions may be formed from multiple conditions using logical operators. Use parentheses to group and nest conditions up to eight levels deep.

AND	logical and	both A and B
OR	logical OR	either A or B or both
XOR	exclusive OR	either A or B, but not both

Example

```

_answers = 42
_dformat = "bib"
IF ((_answers < 100) AND (_dformat = "bib")) /* True
IF ((_answers < 100) AND (_dformat = "all")) /* False
IF ((_answers < 100) OR (_dformat = "ide"))  /* True
IF ((_answers < 15) OR (_dformat = "ide"))   /* False
IF ((_answers < 100) XOR (_dformat = "bib")) /* FALSE
IF ((_answers < 100) XOR (_dformat = "ide")) /* True

```

Special Operators

\\!	Edit	Edit a command before sending it to an inline host (= > statement).
\\>	L-number	Assign the L-number result of a command to a variable (= > statement).
#	answer count	Retrieve the number of answers from an L-number variable.
EXISTS	file existence	Determine whether a given file exists. EXISTS first checks the User Scripts folder and then Scripts folder. It sets <code>\$_filerror</code> to zero if the file exists and to a nonzero value if the file does not exist.

Syntax Requirements

The STN Express with Discover! script language has few general syntax requirements. Each statement has its own syntax, however.

- More than one statement may appear on the same line, or each statement may be typed on its own line.
- The keywords in the Script Language are not case-sensitive. For example, the EXIT statement may be written EXIT or exit.
- Upper- and lowercase are significant in character strings. For example, "Yes" is different from "yes".
- Blank lines may appear anywhere.
- Blanks and space characters may appear anywhere. However, they are significant in strings (inside double quotes). For example, "STNExpress" is different from "STN Express".
- A line in a script is limited to 80 characters, but a single script statement can be 140 characters long. To continue a statement on the next line, put a backslash, \, at the end of each line that is continued on the next line.
- Enclose strings in double quotes.
- Comments are notes that describe or explain the script, and they are ignored by the script processor. Comments begin with * and continue until the end of the line. Comments may appear on lines of their own, or they may appear at the end of statements.
- STN commands can be up to 256 characters in length.

Statements

How Statements are Described

The Script Language is composed of “keywords” and “values.” Keywords are words that have special meaning in the Script Language. Values are pieces of data in the form of variables, strings, numbers, or file names. A “statement” has at least one keyword, and some statements have parameters. A “parameter” is the object of the statement and may be a keyword, a value, or a combination of keywords and values. You can sometimes think of statements as having verbs and objects. For example, in the statement `GET _author`, `GET` is the verb and `_author` is the object (parameter).

The conventions used to describe the STN Express with *Discover!* Script Language are:

1. **Keywords** are shown in uppercase letters, e.g., `EXIT`.
2. **Strings** are shown in lowercase letters inside double quotation marks. For example, in the statement `SEND "string"`, `SEND` is a keyword, and `string` is a string value. You may use a variable wherever you can use a string.
3. **Variables** are shown in lowercase letters with a preceding underscore. For example, in `GET _var1`, `GET` is a keyword and `_var1` represents the name of a variable.
4. **Numbers** are denoted by lowercase letters. For example, in the statement `PAUSE n`, `n` represents a number.
3. **Optional parameters** are shown inside square brackets. For example, `PAUSE n [SECONDS]` means the `PAUSE` statement has a required parameter, `n`, and an optional keyword, `SECONDS`. Do not type the brackets when you type the parameter or keyword.
4. When there is a **choice of parameters** or values, the choices are enclosed in braces, and a vertical bar separates the choices. For example, in the statement `CAPTURE {ON | OFF}`, the keyword `ON` or the keyword `OFF` follows the `CAPTURE` keyword.
5. **File names** are shown within angle brackets, for example, `DELETE <file name>`. You may use a variable whose value is a file name wherever you can use a file name.
7. **Comments** are preceded by `*` and continue through the ends of their lines. The script processor ignores comments.
8. **Examples** are shown in a fixed-width font.
9. **Script output and results** are shown in a **bold font**.
10. Note that a **block** is a single statement or a group of statements enclosed by `BEGIN` and `END`.

The statements in the STN Express with *Discover!* Script Language are listed in alphabetical order.

- Make sure you review the common statements `SEND`, `WAIT`, `=>`, `:`, `ECHO`, `IF`, `GOTO`, and `EXIT`.

=>
:

=> [host-command] [!] [\> _var] (primary prompt statement)
: [STN-reply] [!] [\> _var] (STN secondary prompt statement)

The => and : statements are shortcuts for a WAIT/SEND combination. The => command waits for a primary or level 1 prompt, “=>” on STN. The : command waits for a secondary or level 2 prompt, “:” on STN. After the prompt is received, **host-command** or **STN-reply** is sent to the online host. For example:

```
=> del history          \* Send a delete history
                        \* command
: y                    \* and reply to the secondary
                        \* prompt.
```

is equivalent to:

```
SEND ""                \* Cause an arrow prompt.
WAIT
FOR "=>"              \* Wait for the arrow prompt
SEND "del history"    \* and send a delete command.
WAIT
FOR ":"               \* Wait for the confirmation
                        \* question
SEND "y"              \* and reply.
```

In each case in the example, STN Express waits to receive a command prompt from STN. After receiving the arrow prompt, STN Express sends a delete history command and then waits for a secondary prompt from STN. STN responds with “DELETE ALL L# ITEMS? (Y)/N:”, which STN Express recognizes as a level 2 prompt

The => statement works for online hosts other than STN. The : statement is for STN only.

Use the \! option if you want to edit the host command before it is sent (see also the EDIT statement).

Use the \> operator to assign the L-number result of the command to a variable. For example, suppose the search command in the following statement produces L2 and 1492 answers:

```
=> s acid cow \> _lnum
```

The value of _lnum will be “L2”. Furthermore, the # operator can be used with _lnum to retrieve the number of answers in an L-number. The statement:

```
ECHO "_lnum contains #_lnum answers."
```

displays:

L2 contains 1492 answers.

Assignment Statement `_var = expression`

Use an assignment statement to “assign” a value to a variable. First type the name of the variable, then an equal sign (assignment operator), and an expression. An expression can be a variable, a string, a number, a file name, an arithmetic expression, or a function. For example:

```
_count = 0
_count = _count + 1
_author = "dittmar, p"
_filename = <rn.txt>
```

BEGIN/END BEGIN/END

BEGIN/END forms a *block* of statements that are treated as a group. BEGIN may appear on the same line as an IF statement.

A BEGIN/END block usually follows an IF or ONEXIT statement.

To use this statement, type BEGIN before the statements you want grouped together. Type END after the statement group.

For example:

```
IF (_status = "NOT CONNECTED") BEGIN
    ECHO "DIALING..."
    SEND "ATDT 555-5555"
END
```

This example executes every line between BEGIN and END only if the value of `_status` is “NOT CONNECTED”. Otherwise, the script continues after the END statement.

BREAK BREAK [n [MILSEC[S]]]

The BREAK statement sends a break to the host system. Enter the number, `n`, as either a variable or an integer. If `n` is not entered, 750 is the assumed value.

For example:

```
BREAK 500
```

sends a break 500 milliseconds in duration.

CAPTURE**CAPTURE {ON | OFF} <[file name [/A]]>**

The CAPTURE statement turns on or off transcript capture for your online session.

If the CAPTURE ON file already exists, it is overwritten. /A opens the transcript in append mode.

Entering a file name is optional. If you do not enter a file name, you must still type <>. CAPTURE ON <> prompts you to enter the transcript file name using the standard Capture Session dialog. The Capture Session dialog allows you to overwrite or append to an existing transcript.

Capture may be to a standard STN Express transcript or to an RTF file. To capture to an RTF file, include a ".rtf" file name extension. The use of any other file name extension, or no extension at all, results in a standard .trn-format transcript.

CAPTURE OFF <> stops capture.

For example:

```
CAPTURE ON <tranfile>
ECHO "THIS LINE OF TEXT"
=> e smith/au
CAPTURE OFF <>
```

results in the line "This LINE OF TEXT" and the result of the expand command being placed in a file named tranfile in the Transcripts folder defined in your General Preferences.

Capture is turned off when STN Express with *Discover!* receives the CAPTURE OFF <> statement or when the online session ends.

If an error occurs, the _\$filerror system variable is non-zero.

CLOSE**CLOSE**

The CLOSE statement closes the currently open file.

For example:

```
CLOSE
```

closes the data file that was opened by the OPEN statement.

If an error occurs at the end of the file, the _\$filerror system variable is non-zero.

CONTINUE

CONTINUE

The CONTINUE statement is used to continue processing a script beyond the current SEND, WAIT FOR, and TRYAGAIN statements.

For example:

```
SEND "X"  
WAIT  
  FOR "LOGINID" CONTINUE  
  FOR 20 SECONDS GOTO @ERR  
SEND _loginid
```

If "LOGINID" is received from the host, the script continues at the SEND statement.

DELETE

DELETE <file name>

The DELETE statement deletes a specific file.

For example:

```
_filename = <oldfile.txt>  
DELETE _filename
```

deletes the file named OLDFILE.txt.

If an error occurs, the _\$filerror system variable is non-zero.

DROP DTR

DROP DTR

The DROP DTR statement is used to set the DTR (Data Terminal Ready) circuit to an "off" state. In some rare situations, you may need to control the DTR setting on your modem. See also RAISE DTR.

ECHO**ECHO “string” [NOCR]**

The ECHO statement is used to display a character string or variable on the screen. The “string” can be zero or more characters, variables, or both. A carriage return is included after the string. To suppress it, use the NOCR keyword. For example:

```
_var = "example"
_item = "ECHO statement."
ECHO "This is an _var of the " NOCR
ECHO _item
```

displays

This is an example of the ECHO statement.

on your screen.

The ECHO statement is not used to send commands to an online host (use the SEND statement). The results of ECHO statements and host commands are mixed in the session window and transcript. As a result, it can appear that commands are being sent to the host when they are not. For example,

When you use the ECHO statement in conjunction with the GET statement, echo a blank line first (carriage return only). Then ECHO a prompt or message.

For example:

```
ECHO "Enter your search term:"
```

appears as

=> Enter your search term:

on STN. You may wish to use an extra ECHO to avoid this. For example,

```
ECHO "" \*outputs a blank line to the screen
ECHO "Enter your search term:"
```

Appears as

=>

Enter your search term:

In the STN session and transcript.

EDIT**EDIT {ON | OFF}**

The EDIT statement is used to interact with the script processor by pausing the script before sending a command to the online host. An "Enter user data" box with the next command in it appears, allowing the command to be edited before it is sent. The changes made in the text editing box are not saved in the script file. The default is EDIT OFF.

For example:

```
=> FILE CAPLUS  \* This line is sent as typed
EDIT ON
=> S ASPIRIN    \* This line can be modified
               \* before it is sent

EDIT OFF
=> D 1-3 ALL    \* This line is sent as-is
```

You may use \! as a shortcut for the EDIT statement. Use \! on the => and : statement lines for STN scripts. This shortcut must occur at the end of the statement.

EXEC**EXEC <file name>**

The EXEC statement is used to execute another script from within a script. The script file name is required.

Scripts may be executed eight levels deep. Variables defined in the calling script are also available for the called script. When the called script completes, processing continues on the line immediately after the EXEC statement in the calling script.

For example:

```
_filevar = <SECONDSSCRIPT.sc>
EXEC _filevar
ECHO "_filevar has completed processing."
```

executes the script SECONDSSCRIPT.sc. Then it returns to the calling script and prints:

SECONDSSCRIPT has completed processing

All scripts that are executed must reside in the User Scripts or Predefined Scripts folder as defined in General Preferences (located in the Setup menu on the Main Menu). The User Scripts folder is checked first for the script. If it is not found, the Predefined Scripts folder is checked. If the script is not found in either folder, an error message is displayed.

If an error occurs, the `$_filerror` system variable is non-zero.

EXIT

EXIT

The EXIT statement defines a point to leave the script.

An EXIT is assumed after the last statement in a script. Thus, an EXIT statement is not required in any script. More than one EXIT statement may be used in a script.

If an ONEXIT statement exists in a script, the ONEXIT block is executed before exiting. If the ONEXIT statement includes an EXIT statement, an exit occurs immediately when that EXIT is executed.

GET**GET _varname [HIDDEN] [LABEL="string"]**

The GET statement is used to accept input from the keyboard.

When a GET statement is reached in the script, text entry box is displayed in which the user may type. Typed characters, up to but not including the first carriage return, are placed in _varname.

All input is scanned for a colon character (:). If found, GET returns only the characters that follow the colon.

If the HIDDEN keyword is present, the typed characters are not displayed in the input box.

If both the HIDDEN and LABEL keywords are present, HIDDEN must be first.

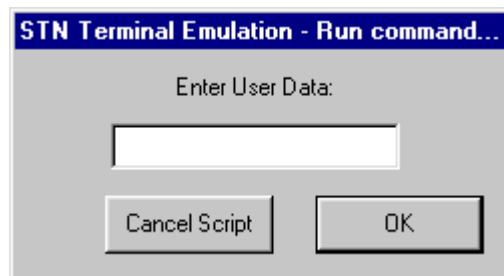
For example, the statements:

```
ECHO "Please enter your password:"  
GET _passwd HIDDEN
```

display

Please enter your password:

In the session window and prompts for your password with an Enter User Data pop-up box:

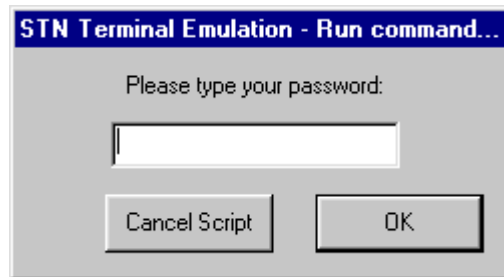


When your password is entered, it is stored in the _passwd variable but not displayed.

Use the LABEL keyword to specify your own message in the pop-up box. For example:

```
GET _passwd HIDDEN LABEL="Please type your password:"
```

causes the following dialog box to be displayed:



GOSUB

GOSUB @LABEL

The GOSUB statement is used to branch to a different location (a subroutine) in the script with the option of returning to the statement that follows GOSUB. Subroutines are useful when an identical set of statements is executed from various points in the script.

GOSUB has only one parameter, a label. When GOSUB is reached, script processing moves to the indicated label and continues processing until a RETURN statement is reached. The RETURN statement causes the script to continue with the statement following the GOSUB statement.

GOSUBs may be nested up to eight levels.

For example:

```
ECHO "Right before the GOSUB."
GOSUB @SUBROUTINE
ECHO "Right after the GOSUB."
EXIT
@SUBROUTINE
ECHO "Within the GOSUB."
RETURN
```

Displays:

```
Right before the GOSUB.
Within the GOSUB.
Right after the GOSUB.
```

GOTO**GOTO @LABEL**

The GOTO statement is used to direct script processing at a line that is not directly after the current line of a script.

GOTO is similar to GOSUB except that there is no option to RETURN.

For example:

```
IF (_var1 = "FIRST") GOTO @LABEL1
ECHO "Value of var1 is not FIRST."
EXIT
@LABEL1

ECHO "Value of var1 is FIRST"
EXIT
```

outputs only one of the above ECHOed lines before exiting depending on the value of `_var1`.

IF**IF (condition) block [ELSE block]**

The IF statement is used to execute one or more statements based on a condition or criteria.

The IF/ELSE statement forms a construct that decides which statement to execute next. If the condition is true, the statement or block directly following the IF is executed. If the condition is false, the statement or block directly following the ELSE keyword is executed. If the condition is false and there is no ELSE keyword, the statement following the end of the IF block is executed.

For example:

```
\* If no phone number was given, ask for it. Otherwise,
\* display the number. In either case, dial the number.
IF (_phoneno = "") BEGIN    \* multi-statement block
    ECHO "Enter phone number:"
    GET _phoneno
    END
ELSE
    ECHO "Dialing _phoneno"    \* single-statement block
SEND "ATDT _phoneno"
```

Note that the SEND statement is outside of the IF/ELSE construct. It is executed regardless of the condition.

ONEXIT

ONEXIT block

ONEXIT is used to specify a set of statements to execute before leaving a script.

The ONEXIT statement may be placed anywhere within a script. An ONEXIT block cannot contain a GOTO or RETURN statement.

When the script processor reaches an EXIT statement and an ONEXIT block has been defined, processing jumps to the ONEXIT block. After the ONEXIT block has been completed, the script ends. An EXIT statement within an ONEXIT block causes an immediate exit (the ONEXIT block is not executed again).

The block portion of the statement may be entered as one line, or as a set of statements between BEGIN and END.

For example, the statements:

```
ONEXIT
  BEGIN
    ECHO "These two lines will always be"
    ECHO "executed upon exiting the script."
  END
```

cause two lines to be displayed on the screen when you leave the script.

OPEN **OPEN <file name> [{/R | /W}]**

The OPEN statement is used to open a new or existing file from within a script.

The file names is optional. If no file name is supplied, you are prompted to enter one.

The /R option opens the file to read only. The /W option allows both reading and writing in the file. The default option is /W.

All data written to the file is appended to the end of the file. To rewrite a file completely, delete it before opening it.

For example:

```
_error = "Command timed out. Error ##36."
DELETE <errors.txt>
OPEN <errors.txt>
WRITE _error
CLOSE
```

places the following message in a file named *errors.txt*:

Command timed out. Error #36.

If an error occurs, the `$_filerror` system variable is non-zero.

PAGE **PAGE**

The PAGE statement forces the captured transcript to be a new page.

Page breaks appear when Browsing a standard transcript as

<-----Page Break----->

and are preserved when saved as RTF.

PAUSE **PAUSE {n [SECOND[S]] | [UNTIL time]}**

Use the PAUSE statement to halt script processing for a period of time or until a specific time of day.

Enter the number of seconds, n, as an integer or a variable. If n equals zero, the pause is forever.

The UNTIL time option allows you to specify a certain time of the day, using a 24-hour clock.

For example:

```
PAUSE 1 SECOND           \* pause for 1 second
PAUSE UNTIL 14:45       \* pause until 2:45 pm
```

PRINT **PRINT {ON | OFF}**

Use the PRINT statement to turn “slave” printing on or off. Slave printing causes the transcript to be printed as the online session progresses. Slave printing is not recommended.

RAISE DTR **RAISE DTR**

Use the RAISE DTR statement to set the DTR (Data Terminal Ready) circuit to an “on” state. See also DROP DTR.

READ **READ _varname**

The READ statement reads a single line from the OPEN file and stores the value, without a carriage return character, in _varname. For example,

```
ECHO "reading first line of INPUT FILE"
OPEN <input.txt>
READ _line
CLOSE
```

The file input.txt opened and the first line is read into the variable _line.

If an end-of-file condition is reached or an error occurs, the _\$filerror system variable will be non-zero.

RETURN**RETURN**

The RETURN statement is used to return the point from which a subroutine was invoked by a GOSUB statement. For example,

```
GOSUB @SUBROUTINE
ECHO "processing will return to this point."
EXIT

@SUBROUTINE
ECHO "After printing this line..."
RETURN
```

displays the following:

```
After printing this line...
processing will return to this point.
```

SEND**SEND "string" [EDIT] [HIDDEN] [NOCR]**

The SEND statement is used to send a string to the online host. (Technically, the string is sent to the communication port specified in the logon setup. Therefore, for example, a string can be sent to a modem before STN Express is connected to the host.)

The string and other parameters may occur in any order after the SEND statement. The "=>" and ":" statements are specific forms of the SEND statement.

EDIT is an option that allows the string to be edited before it is sent. EDIT works as \! does in the => and : statements.

HIDDEN does not display the string as it is sent. This is useful when sending passwords.

A carriage return is automatically included after the string. To suppress it, use the NOCR keyword.

For example:

```
SEND "+++" NOCR
SEND "ATDT 555-1212"
```

sends:

```
+++ATDT 555-1212
```

to the modem.

STNLOGON STNLOGON

The STNLOGON statement is used to execute the STNLOGON script that logs you on to STN International. STNLOGON assumes that there is already a connection to STN. Specifically, STNLOGON sends a carriage return and expects the reply to be "Welcome to STN International! Enter x:".

SUBSTR SUBSTR (_var, begin, length)

SUBSTR a function that takes a string of characters in a variable and finds a "substring" or portion of the string. The returned substring begins with character number begin and continues for a length characters. Begin and length maybe literal numbers or variables. For example, the statements

```
_var1 = "xxx remove this portion xxx"
_var2 = SUBSTR(_var1,5,19)
ECHO _var2
```

display the following contents of _var2:

remove this portion

SUBSTR may be used in conditions. For example:

```
IF (SUBSTR(_var1,1,3) = "xxx") GOTO @label
```

When the script runs, if begin is beyond the end of the string, a variable substitution error is given.

TRYAGAIN **TRYAGAIN n [TIMES] [[THEN] block]**

The TRYAGAIN statement is part of a WAIT block and is used to repeat the previous SEND statement a specified number of times.

This statement works only in the context of the SEND-WAIT statement pair. It repeats the previous SEND up to n times. After the nth attempt, processing either performs the THEN block or EXITS.

For example:

```
SEND "ATDT 555-1212"
WAIT
  FOR "CONNECT" CONTINUE
  FOR "NO CARRIER" EXIT
  FOR 5 SECONDS
  TRYAGAIN 5 TIMES THEN GOTO @LABEL
```

This sends the phone number and waits 5 seconds for either CONNECT or NO CARRIER. If the response is not received, the phone number is sent up to four more times. If the response is still not received, the process jumps to the @LABEL label.

UPLOAD **UPLOAD [LNUM _var1,_var2,_var3...] <file name>**

The UPLOAD statement is used to upload a structure query to STN. The result of the UPLOAD on STN is one or more L-numbered queries.

<file name> is a full path to a .str query file created in Structure Drawing.

The LNUM keyword indicates the beginning of a list of variables. The variables are assigned the L-number or L-numbers of the uploaded structure. Usually, only one variable is needed. Multiple variables are needed when a reaction query with multiple participants is uploaded in an STN file that does not support reaction searching. In that situation, each of the participants is uploaded into its own L-numbered query. Commas must separate variables in the list.

For example:

```
UPLOAD LNUM _line1 <C:\STNEXP\Queries\STR1.STR>
```

This statement uploads the structure query file STR1.STR to STN and stores the resulting L-number value in the _line1 variable.

If an error occurs, the _\$filerror system variable is non-zero.

USER**USER**

The USER statement is used to temporarily halt script processing and give control of the online session to the user. Control is returned to the script when you press the END key.

VT320**VT320 {ON | OFF}**

The VT320 statement is used to turn ON or OFF VT320 terminal emulation and is useful when you connect to STN via a VAX with a menu system and when you connect to an online host that offers a menu system.

WAIT**WAIT [_var1] [LNUM _var2] for-list**

The WAIT statement is used to pause script processing until a specific response is received from the online host. This statement results in data being read from the incoming communication port one line at a time. A line includes everything up to a carriage return.

After WAIT is executed, _var1 contains the last line received from the host.

LNUM _var2 puts an L-number into the variable _var2.

for-list is one or more:

FOR for-condition [THEN] block

This is defined as: "If for-condition is met, then execute these statements."

The block is optional, except for the last for-condition. Often the block contains a CONTINUE statement.

For-condition is either:

"string" (substring received from host)

or:

n [SECOND[S]] (passing of time, n, can be an integer or a variable)

If n is zero, the WAIT is forever.

See the example with the TRYAGAIN statement.

WRITE

WRITE "string" [NOCR]

The WRITE statement writes a line of text into the currently open file.

WRITE places the string in the open file, followed by a carriage return. If the NOCR keyword is present, the carriage return is suppressed.

Lines are always written to the end of the file. To erase the contents of the file before writing to it, use the DELETE statement.

If an error occurs, the `$_filerror` system variable is non-zero.

Sources of Example Scripts

Use the following scripts as models for scripts that meet your needs. To see other scripts, look in the Scripts and PSSfiles folders in your STN Express installation.

Script Tips

- Scripts do not require a specific format or style as you type the statements, and more than one statement may appear on the same line. However, for ease of understanding the scripts you have written, use a style similar to what you see here.
- To ensure that all STN Express features are available while you are online, use the STNLOGON script instead of writing your own commands to log on to STN.
- Open a new script file by selecting New from the File menu within the STN Edit window. A new, Untitled window is displayed

- Check Command File checks your script for some errors. Getting an error-free script from the command file checker does not guarantee that the script will do what it was intended to do. A clean bill of health from the command file checker also does not guarantee there are no errors in the script because some errors cannot be found until the script is run.
- Add comments to a script by placing a `*` anywhere on the line. To display a comment on the screen, use the ECHO statement.
- Use comments to help you and others understand it.
- Use blank lines to make your scripts more readable.
- Use a standard spacing and indentation scheme to make your scripts more readable.
- When running a script, press the Escape or Clear key to stop the script.
- All STN Express with Discover! scripts must be in either the User Scripts folder or the Predefined Scripts folder. When looking for a script to execute, STN Express with Discover! first checks the User Scripts folder, then the Predefined Scripts folder.
- Enter only the unique portion of the prompt when using the WAIT FOR statement.
- Use the SEND, WAIT FOR, and TRYAGAIN statements together because TRYAGAIN will resend the last item that was sent using a SEND statement. Many times, TRYAGAIN is an optional part of the statement trio because you would not want to resend some items, e.g., login ID or password.
- You may use your own word processor to write a script as long as you save the file in a text-only format.

Registry Number Script Example

Recall the script shown at the beginning of this document:

```
\* Display CAS RNs in the Registry file on STN.
=> FILE REGISTRY
=> DIS 50-00-0
=> DIS 7777-77-7
=> LOG Y
```

It does its job but is not very flexible. For example, what if you have long list of CAS Registry Numbers instead of just a few? Use the OPEN, READ, IF, and CLOSE statements to process a file of Registry Numbers.

```
\* STN Express script to read list of RNs from a file
\* and display information for each Registry Number.

\* Open the data file for reading. There is one RN on
\* each line.
OPEN <rnlist.txt>/R

=> file registry

\* Process all the RNs in the file
@readloop

\* Read an RN from the data file.
READ _rn

\* If an RN was read, send a DISPLAY command to STN.
IF (_$filerror = 0)
  BEGIN
    \* Display the requested substance
    => dis _rn
    GOTO @readloop
  END

\* Close the input file
CLOSE
=>
EXIT
```

Author Search Script Example

```

ECHO ""
ECHO "This is a script for running author searches."
ECHO "Press [Esc] at any time if you would like to\
stop."
ECHO ""

ECHO "In the pop-up box, enter the last or family\
name of the author:"
GET _lastname
ECHO "In the pop-up box, enter first or given name\
or initial:"
GET _firstname

\* Use the STN expand command to provide choices of
\* names.
=>
=> expand _lastname _firstname/au

ECHO "Enter the e-numbers to search, e.g., e3-e5 or\
e3,e7) or end"
GET _enums
IF (_enums = "end")
  BEGIN
    ECHO "Ending author search at your request."
    =>
    EXIT
  END
=>
=> search _enums \> _results

\* If the search yielded zero answers, there is
\* nothing to display. Inform the user and exit.
IF (#_results = 0)
  BEGIN
    ECHO "There are no answers for your query."
    =>
    EXIT
  END

```

```
\* The search found hits, so ask how many should be
\* displayed.
ECHO "How many answers would you like to display? (e.g.,
\#_results or 0):"
GET _display

IF (_display > 0)
  BEGIN
    ECHO "Please provide a transcript name to use to\
    save your answers."
    CAPTURE ON <>

    \* Don't try to display more answers than are
    \* available.
    IF (_display > #_results) THEN _display = #_results

    =>
    => dis 1-_display bib
    CAPTURE OFF <>
  END

ECHO "Your author search is now complete."
=>
EXIT
```

Logon Scripts

There are two separate steps to logging on to an online host:

1. Connecting to the host through a telecommunications network.
2. Logging on to the host.

STN Express includes support for several telecommunications networks and online hosts. Before you take the time to write your own connection or logon script, find out whether STN Express already provides the scripts you need. Also, you may find it easier to use the Watchme feature than to write your own logon script.

For more information about logon setups, including Watch Me, visit STN Express Support at www.cas.org.

Predefined Logon Setup Variables

If you intend to write your own scripts that connect through a network or log on to an online host, be sure you understand the details of a logon setup. Many of the values defined in a logon setup are available to STN Express with *Discover!* scripts as variables. Below is a list of these variables, organized by the area within Setup that sets the value.

Setup Definition Dialog

- Host Information

_\$LOGID	Login ID
_\$PASSWORD	Password
_\$SERVICE	Host Name; set to "STN" for all STN hosts
_\$STNCENTER	Name of STN Node (Columbus, Karlsruhe, or Tokyo)
- Path

_\$CONNVIA	Connect via
_\$NETWORK	Value selected from network list when the Logon Method is Standard
_\$NUA	NUA (Network User Address)
_\$NUI	NUI (Network User Identifier)
_\$NUP	NUP (Network User Password)
- Communication Settings

_\$BAUD	Speed
_\$DIALTYPE	Dial
_\$PHONENO	Primary Phone #
_\$PHONENO2	Secondary Phone #
- Host Settings tab

_\$QSERVNAME	Choice of service (Questel)
_\$GRAPHICS	Graphics (STN)
_\$STNPORT	STN Port (STN)

Advanced Dialog

- **Prompt Character strings**
 - _\$GWLOGIN** Login
 - _\$GWCOMMP** Command
 - _\$GWPASSP** Password
- Execute at command prompt
 - _\$GWCOMM1** Execute at command prompt field one
 - ...
 - _\$GWCOMM5** Execute at command prompt field five
- Optional
 - _\$GWID** Gateway Login ID
 - _\$GWPW** Gateway Password
- Modem Configuration
 - _\$MODINIT** Initialization String

Predefined STN Variables

In addition to predefined logon setup variables, other variables related to your STN session are available to you.

_\$CENV	The current STN file environment (a list of upper-case STN file names)
_\$ENUM	The current E-number
_\$LANS	Number of answers in the most recent L-number
_\$LNUM	The most recent L-number
_\$LENV	The STN file environment on entry to the script

Trademarks

STN, STN Express, CASREACT, and MARPAT are registered trademarks of the American Chemical Society. SPECINFO is a trademark of Chemical Concepts GmbH. ChemDraw is a trademark of Cambridge Scientific Computing, Inc. DARC, Questel, and Orbit are registered trademarks of Questel. S.A.HPGL is a registered trademark of Hewlett-Packard Co. SMILES is a trademark of Daylight Chemical Information Systems, Inc. NEC is a registered trademark of Nippon Electronics Corp. Tektronix is a registered trademark of Tektronix, Inc. MDL and ISIS are trademarks of MDL Information Systems, Inc. IBM is a registered trademark of International Business Machines Corp. DataStar and DIALOG are registered trademarks of Knight-Ridder Information, Inc. Ovid is a registered trademark of DCP Technologies, Inc. Windows is a registered trademark of the Microsoft Corp. Adobe and Acrobat are registered trademarks of Adobe Systems, Inc. BLAST is a registered trademark of the National Library of Medicine. QuickTime and the QuickTime logo trademarks used under license.