MySabre Scribe

SCRIPTING GUIDE

Table of Contents

Edition 1.2 (December 2007)

Introduction

Welcome to *MySabre*® *Scribe*SM software! *MySabre Scribe* software is a PC-based scripting development software package. It allows you to automate and customize *Sabre*® computer reservations system processes through the development of scripts.

Who should Use This Guide

This manual is intended as a guide for anyone planning to build scripts and is designed for agents who have attended the *MySabre Scribe* Basic class.

This guide, which complements the class materials, will help you develop the skills necessary to design and write scripts. It assumes you are familiar with:

- Basic *Sabre* system formats
- Basic travel industry terminology and concepts

Whether you are an existing script developer or a newcomer to the scripting world, you should soon be able to create basic or sophisticated scripts for your office through the use of *MySabre Scribe* software.

MySabre Scribe software is designed for use by people without a programming background; however, programming experience is recommended for developing complex scripts.

Purpose of this Guide

This guide provides information to help you design and develop scripts. It contains details about the basics of *MySabre Scribe* software, the scripting language, using the developer, and using the compiler. It also provides guidelines and helpful hints to assist you in developing efficient scripts.

After attending the *MySabre Scribe* Basic class, you will find this guide a useful tool when building scripts. As your skills increase, you can expect to apply more and more elements contained in this manual.

How this Guide Is Organized

The remainder of this guide includes

- Understanding *MySabre Scribe*
  - Provides an overview of *MySabre Scribe* software and describes recent enhancements and new features you can use to make your scripts work best for your agency.
- Scripting Basics
  - Describes basic elements of scripts, such as variables, text strings, and expressions.
- Using the Developer
  - Provides an overview of the Developer, including how to start it and use it to create a script. Also provides an overview of how to compile scripts, including details about viewing error messages.
- Advanced Scripting
  - Describes advanced elements of scripts, such as subroutines, chains, spawns, reading, and writing.
- Techniques for Developing Scripts
  - Provides techniques and hints for creating powerful, efficient scripts.
- Glossary of Terms
- Index

Related Documentation and Training

In addition to this Scripting Guide, Sabre Inc. has a variety of information and training available:

- *MySabre Scribe* Scripting Reference Guide, which provides details about the commands and other constructs of the scripting language.
- *MySabre Scribe* Installation Guide, which helps you install MySabre Scribe software.
- *MySabre Scribe* Basic class

All this documentation and training complements the information provided in your *MySabre Scribe* Basic classroom training.

Understanding *MySabre Scribe*

MySabre Scribe Software Components

MySabre Scribe software consists of three components:

- MySabre Scribe Developer, which you use to write the source code for your script.
- *MySabre Scribe* compiler, which you use to convert the source code to executable code, which can operate on the workstations in your office.
- *MySabre Scribe* run time, which the agents in your office use to execute the script you created. Several sample scripts are included which you may copy and modify to your special needs.

MySabre Scribe Features

MySabre Scribe features include:

- Ability to use *MySabre Scribe Developer* enhancements with existing *Scribe* compiler
- Compatible with existing Scripts
- Ability to prepare an email shell from *MySabre Scribe Developer* script
- Capacity to launch third party application (for example, Notepad) from *MySabre Scribe Developer* script
- Ability to read from and write to and from .CSV files
- Ability to run a script with name longer than 8 characters
- Correct display of *Sabre* system special characters in menus and windows
- Syntax Highlighting
- Syntax Error Reporting
- Global Variables
- Auto Complete
- Buttons to input *Sabre* system special characters
- EMUFIND provides the ability to parse the emulator screen using strings, regular expressions, or number of characters to skip.

Benefits of Using *MySabre Scribe*

Developing and using scripts can benefit your agency in the following ways:

- Walk agents through the complicated steps of hotel bookings to avoid later quality control problems.
- Maintain consistency in the reservation process of booking air, car, and hotel reservations.
- Automatically prompt agents to record special meals or special airfares for each client.

Automating *Sabre* system functions and manual processes through scripts is an ideal way to save valuable agent time. It can also improve quality control, increase efficiency, and streamline procedures. Efficiently designed scripts reduce the time needed to process transactions and improve the quality of the transactions.

This makes work easier for both new and experienced agents. You can reduce the amount of training needed for new employees by standardizing processes in your agency with scripts.

Scripting Basics Overview

This section provides details about the basic constructs of scripts. It provides an overview of scripts along with details about key elements of scripts:

- Character strings
- *MySabre Scribe* software commands
- Sabre system commands
- Expressions
- Variables

It also provides details about elements that can enhance your scripts—remarks and labels.

Script Overview

A script is an ASCII file with statements that the *MySabre Scribe* compiler understands. You create the ASCII file using the *MySabre Scribe Editor* and add statements to it.

After adding statements to your script, you use the *MySabre Scribe* compiler to create an executable file for *MySabre Scribe* run time. When a user wants to use a script, the user sees the executable file and starts it. This causes run time to execute the statements you included in your script.

Statements are read from top to bottom, left to right. You put statements in a specific order denoting a logical flow. Statements generally appear in uppercase. You may use lowercase in some statements. The *MySabre Scribe* Scripting Reference indicates when lowercase is acceptable.

The compiler recognizes different types of statements based on identifiers you use. The following table highlights the identifiers:

Identifier	Description	Example
""	Quotes indicate character strings	"This is a string."
CAPS	Capital letters indicate a *MySabre Scribe* software command or label	EXIT
:	A colon following capital letters indicates a label name	BEGINNING:
»«	Chevrons indicate *Sabre* system commands	»*A«
[]	Brackets indicate a variable	[VARIABLE1]
;	Semi-colons indicate remarks	; This is a remark
=, <, >, <=, >=, <>, AND, OR	Operators test a condition or assign a value	[A] = [B] + 3

When the compiler finds these identifiers, it expects the identifiers immediately following the characters to adhere to strict rules. This guide and the *MySabre Scribe* Scripting Reference Guide describe these rules.

Using Character Strings

What is a Character String?

A string or character string is any series of characters in quotes. Airport codes, city codes, airline codes, and free form text are all examples of strings. A string can consist of numbers, letters, punctuation, or any other valid ASCII symbol as long as you enclose it in quotes.

MySabre Scribe software allows you to add character strings. For example: "Sab" + "re" equals "Sabre".

If you enter numeric data or a numeric expression enclosed in quotes, no calculations can be performed on the data in the string.

Example:

```
DEFINE [TOTAL_S = ALPHA:10]
DEFINE [TOTAL_N = NUMERIC:10]
[TOTAL_S] = "10"
[TOTAL_N] = 10

[TOTAL_N] = [TOTAL_N] + 10
[TOTAL_S] = [TOTAL_S] + "10"
```

In the previous example, the variable [TOTAL_N] contains the value 20 after execution because it was acted upon as a numeric value rather than as a string. The variable [TOTAL_S] contains the value 1010 after execution, because the string value is appended to the original string value of 10.

You can embed quotes inside strings by inserting a backslash (\) before the quotes. The compiler translates the quote after the backslash as a single quote. This adds flexibility to the text you can provide in your scripts.

Example

```
.
.
.
DEFAULT [NAME] = "JOHN \"Jack\" KENNEDY"
WINDOW
The passenger's name is [NAME]
ENDWINDOW
.
.
.
```

The window displays with the following text: The passenger's name is JOHN "Jack" KENNEDY

Using MySabre Scribe Commands

MySabre Scribe software commands are keywords that the compiler recognizes. They appear in uppercase. The compiler processes these commands in a specific way depending on the command. For example, the compiler knows that the EXIT command stops the processing of the active script and returns the user to the *Sabre* system.

Most commands have *parameters*. Parameters are values the compiler associates with the command.

The parameters the compiler expects differ depending on the specific command. The compiler may expect a number, a string, or an identifier, such as a label. The compiler may accept an expression that evaluates to one of these values.

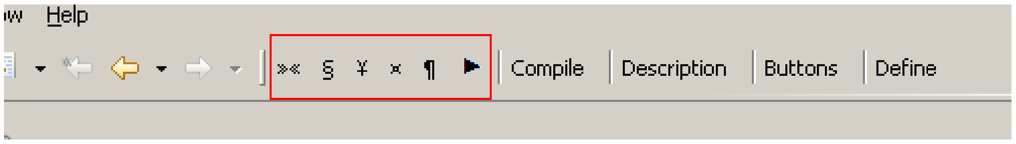
Some parameters are required; some are optional. The *MySabre Scribe* Scripting Reference indicates which parameters are required and optional.

Accessing Sabre System Commands

A *Sabre* system command statement sends a format to the *Sabre* system. These statements:

- Begin with a chevron “»” and end with a chevron “«”
- Should be on a separate line in your script

To insert chevrons, click the chevron button on the Special Character pallet.



What is an Expression?

An expression is a string of characters that the compiler evaluates. An expression is a mechanism for combining several pieces of data. The data can be strings, values, variables, or other expressions.

The compiler uses the operators to determine how to evaluate an expression. The compiler uses the following order of precedence to evaluate operators. If several operators of the same precedence appear on the same line, they have equal precedence.

1. Parentheses: ()
2. Multiplication and division: *, /
3. Addition and subtraction: +, -
4. Comparison: <, <=, =, >=, >, <>
5. Logical: and, or

   Example: 3 + 8 * 4 - 2

In the example above, 8 will first be multiplied by 4. The result (32) will then be added to 3. Two will then be subtracted from the result. The compiler calculates the expression in this order because the multiplication operator (*) has precedence over the addition operator (+) and the subtraction operator appears after the addition operator.

MySabre Scribe software provides commands that allow your program to make decisions based on test expressions you specify. The most commonly used command is the IF/THEN.

The IF/THEN command allows you to specify an expression you want tested before executing another command or series of commands. For example, you want to determine if you need a car before you buy it.

In *MySabre Scribe* software, you would represent this statement as follows:

```
IF [CAR_NEEDED] = "Y" THEN

CALL BUY_IT

ENDIF
```

IF is the command used to identify an expression ([CAR_NEEDED]="Y") that you want verified for validity.

The “=” sign is a relational operator used when the software evaluates the test expression. Several commands rely on test expressions that make comparisons of values using relational operators. The supported relational operators are:

EQUALS	=
NOT EQUAL TO	<>
LESS THAN	<
LESS THAN OR EQUAL TO	<=
GREATER THAN	>
GREATER THAN OR EQUAL TO	>=

Relational operators test an expression and return ‘1’ if the expression is true and ‘0’ if the expression is false.

THEN is the command used to identify the action (CALL BUY_IT) you want executed if the test expression evaluates to TRUE.

Test expressions can also use *logical operators.* Logical operators allow you to combine two or more expressions and test them for validity.

AND	Returns “1” for true when each expression evaluates to true. Returns “0” for false when any expression evaluates to false.
OR	Returns “1” for true when any expression evaluates to true. Returns “0” for false when none of the expressions evaluate to true.

The following examples demonstrate the use of the logical operators with the IF/THEN command:

```
IF [FIRST_NAME]="" OR [LAST_NAME]="" THEN
```

Verifies that one of the names is non-blank prior to executing the THEN command.

```
IF ([FIRST_NAME] = "DAVID") AND ([LAST_NAME] = "PETERS") THEN
```

Verifies that both variables are equal required values before executing the THEN command.

In the previous example, each expression appears in parentheses ( ). This notation emphasizes the two conditions tested. This makes it easier to read your script.

Parentheses also specify the order that the software uses to complete a test or a calculation. The compiler uses a specific order when it finds relational and logical operators:

1. Parentheses ()
2. Length and negate $, -
3. Multiplication and division *, /
4. Addition and subtraction +, -
5. Relational operators <, >, =, <>, >=, <=
6. Logical operator AND
7. Logical operator OR

Using Variables

Overview

A variable is a symbol that defines a value. The value could be a number, a character string, or an expression. *MySabre Scribe* software has two types of variables: user-defined and system-defined variables.

These variables can be used by your script only, by all scripts you create, or by specific scripts you create.

User-Defined Variables

The compiler identifies a user-defined variable when it finds brackets. The symbol name assigned to the variable exists between the brackets. Example:
[Variable_name]

When you define a variable, you can control several aspects of the variable by specifying *attributes*. Attributes identify the characteristics of your variable. For example, you could define a variable that must be a character string of less than 30 characters. The *MySabre Scribe* Scripting Reference describes the attributes you can define for variables.

The software allows two types of user-defined variables:

- User-defined variables that you create, using the DEFINE command
  - Variables you add using a DEFINE command are *local* variables. This means they are available only to your script. You can make a local variable available to another script; however, you must identify the variable using the EXTERNAL command in both scripts.
- Global variables
  - Global variables begin with # (pound/hash) character.
    (For more information on global variables, refer to the *MySabre Scribe* Scripting Reference Guide.)

Checking the Number of Characters

MySabre Scribe software provides a tool you can use with variables. If you place dollar sign before a variable (example: **$[VAR1]**), the software tests the number of characters in a variable. Use this tool to validate data before sending it to the *Sabre* system or continuing your script. This will reduce the number of possible errors in your script.

Because $[Variable Name] allows you to measure the length of data in a variable field, you can also control the READ and WRITE commands that interact directly with the *Sabre* system without displaying data on the EMULATOR screen.

This tool does not test the value of the data, only the number of characters in the data.

Relational operators (=, <>, <, >, >=, and <=) are typically used with this tool. The result of the test is returned as a whole number (with no decimal point). You can define another whole number to be compared with the results of the test, determining the next action of the script

Example:

```
DEFINE [CITY=ANY:30:OPT:::CITY CODE]
CITY_CODE:
   WINDOW
   R=1 C=5
   Please Enter the Origin City Code: [CITY]
   ENDWINDOW

IF $[CITY] > 3 THEN
   »W/-CC[CITY]{ENTER}«
   GOTO CITY_CODE
 ENDIF
```

In this example, the variable **[CITY]** is defined with a length of 30. If the agent does not know the three-letter code for the city, the agent can type in the whole name. The script then tests the number of characters in the variable and finds more than three characters.

The format for encoding a city name is executed in the second partition and a GOTO command re-displays the window. The agent can now properly input the three-letter city code before the script continues.

System Variables

The compiler identifies a system-defined variable when it finds an “@” symbol inside brackets. The symbol name assigned to the variable immediately follows the @ symbol. Example: **[@VARIABLE_NAME]**

System variables are pre-defined global variables. This means their attributes are declared by the system and cannot be changed. There are four types of system variables:

- **System resource variables**—The values of these variables are “read only” and cannot be modified by a script. Examples are system time and date.
- **Command button variables**—These variables are used in windows you create in your script. These command buttons appear as pushbuttons and become controls for the window when it is executed in the *Sabre* desktop reservations software. There are six pre-defined command buttons that you can add to your windows: [@OK], [@CANCEL], [@YES], [@NO], [@IGNORE], and [@HELP].

- There are also fifty generic command buttons. The label within these buttons can be assigned by the script:
  - [@BTN0] through [@BTN9]
  - [@BTN_8_0] through [@BTN_8_9]
  - [@BTN_L10_0] through [@BTN_L10_9]
  - [@BTN_L15_0] through [@BTN_L15_9]
  - [@BTN_L20_0] through [@BTN_L20_9]
- The maximum number of characters allowed for each button is specified in its name. For example, [@BTN1] has a length of 5 and [@BTN_L20_4] has a length of 20.
- To invoke command buttons, press the ALT key and the highlighted letter in the button text or press ENTER when the command button is highlighted. To specify a highlighted letter, prefix the desired character with an ampersand (&). The ampersand is not displayed in the button when executed in the run time or test mode. Refer to the WHENEVER/ENDWHENEVER command section for more information about the usage of buttons to display help text for a window.

- **Command execution variables**—This variable stores the execution of a command button in the variable [@BTN_PRSD]. This variable contains the default value of the last button pressed and can be used in any of the comparison commands, such as IF, REPEAT, or WHILE. (Example: IF [@BTN_PRSD] = "&IGNORE" tests the value of a generic command button.) The value is updated with the default value as soon as the button is depressed, and the value is cleared of the default value when a new window is displayed. The value of this system variable allows command buttons to control the entire execution of a script.
  - If you specify a system variable as a command button in your window and the system variable is not specified in a WHENEVER, the window automatically closes when the agent invokes the action of the button and the value of the system variable [@BTN_PRSD] is updated.
- Switches variables—These variables appear with [**@SWITCHES**]=*value*, where value is the setting as defined in each of several classes of switches. These switches control aspects of *MySabre Scribe* run time. The values you specify with this variable remain active throughout run time unless changed, and will affect all Spawned and Chained scripts. There are four different classes of switches. Multiple switches can be set/reset simultaneously. See “Using the @SWITCHES System Variable” for more information.
  - In the following example, a PNR itinerary has been built in the Amtrak system and the window is asking for the rest of the PNR information and displaying buttons with the choices of Merge the Itinerary and Ticket Now or Merge the Itinerary and Ticket at a Later Date. It then either Ignores the PNR or Merges the Itinerary and calls a subroutine that assists in building the *Sabre* system PNR.

Example:

```
 START:
>>*A{ENTER}<<
 [@BTN_L10_1] = "&IGNORE"          ; Button commands
 [@BTN_L15_1] = "&MERGE ONLY"              ; Button commands
 [@BTN_L15_2] = "MERGE &TICKET"            ; Button commands

 WINDOW
 R=1 C=5  H="AMTRAK PNR Displayed"
 As of [@TIME] Today.         ; System resource
 Enter the information requested below:
 Last Name:   [L.NAME]
 First Name:  [F.NAME]
 Home Phone:  [H.PHONE]
 Work Phone:  [W.PHONE]
 Agents Name: [RCVD.FROM]

 [@BTN_L10_1]  [@BTN_L15_1]  [@BTN_L15_2]
 ENDWINDOW

 IF [@BTN_PRSD] = "&IGNORE" THEN          ; Command execution
         >>I{ENTER}<<
         EXIT
 ENDIF



>>-[L.NAME]/[F.NAME]{ENTER}<<
>>9[H.PHONE]{ENTER}<<
 >>9[W.PHONE]{ENTER}<<
 >>6[RCVD.FROM]{ENTER}<<

 IF [@BTN_PRSD] = "&MERGE ONLY"  THEN
     >>E{ENTER}<<
 ELSEIF [@BTN_PRSD] = "MERGE &TICKET" THEN
     >>E¤{ENTER}<<
 ELSE
   GOTO START     ;This would be the case if no
                  ;button was actually pressed in the
                 ;window
ENDIF
```

Using the @SWITCHES System Variable

The fourth type of system variable has several classes that control the

- Display format for TIME
- Clipping of spaces from variables
- Character conversion from DOS to Windows
- Display of the warning message when storing large variables

You can use multiple switches simultaneously. For example, you can set both display format and clip spaces.

Display Format for TIME

You can control how TIME variables are formatted for output by using:

- [@SWITCHES] = "24HOUR"
  Displays TIME in 24-hour format
- [@SWITCHES] = "12HOUR"
  Displays TIME in 12-hour format

MySabre Scribe software defaults to 24HOUR.

Clipping Spaces from Variables

You can control leading spaces and trailing spaces within variables. For example if a variable obtains the value " DAVID ", you can change it to "DAVID". This helps you manipulate the text you use in variables. Use the following to control leading and trailing spaces. The switch only affects variables that are updated after the switch is set.

- [@SWITCHES] = "CLIPALL"
  Removes leading and trailing spaces from variable values
- [@SWITCHES] = "CLIPFRONT"
  Removes only leading spaces from variable values
- [@SWITCHES] = "CLIPBACK"
  Removes only trailing spaces from variable values
- [@SWITCHES] = "NOCLIP"
  Keeps leading/trailing spaces in variable values.

MySabre Scribe software defaults to CLIPALL.

Example:

```
.

.

.

DEFINE [HEADER=ALPHA:40]

[@SWITCHES] = "CLIPALL"

[HEADER] = "     Time Window     "
WINDOW
H=[HEADER]
The current time is [@TIME]
ENDWINDOW
.
.
.
```

This changes [HEADER] to "Time Window" instead of "   Time Window   ", eliminating the leading and trailing blanks.

Warning Messages when Storing Large Variables

You can control whether a warning message appears if a script attempts to store a value into a variable when the value is larger than the variable will contain. Use one of the following:

- [@SWITCHES] = "SHOWTRUNCATE"
  Causes run time to display a warning if the script attempts to store a value larger than the variable will contain. Pressing enter removes the warning and shortens the value to fit the variable.
- [@SWITCHES] = "NOSHOWTRUNCATE"
  Causes run time to shorten the value to fit the variable without displaying the warning.

MySabre Scribe software defaults to SHOWTRUNCATE.

Example:

```
DEFINE [SOME_VAR=*:3]

[SOME_VAR] = "1TEST"
[@SWITCHES] = "NOSHOWTRUNCATE"
[SOME_VAR] = "2TEST"
```

The first assignment displays the warning and the variable will contain “1TE”. The second assignment will not display a warning, and the variable contains “2TE”

Adding Remarks

A remark is any comment you add to explain or clarify the commands and organization of your script. The compiler identifies a remark when is finds a semicolon “;”. After the semi-colon, the compiler ignores any text appearing on the remainder of the line.

Remarks

Appear at the end of a statement or on a separate line.

Enable you to document your script source code. You can add notes about each section of your code explaining the use of certain commands or routines within your script. This simplifies tracing the logic of steps during testing or future modifications.

Using remarks is highly recommended as a method for documenting the logic in your script to facilitate later modifications.

Example:

```
;this remark can explain any unique commands or
; remind you about any special circumstances that
;change with the execution of this menu.

MENU
R=1 C=80 S=0
"BOOK CAR"
  CALL CAR
"BOOK HOTEL"
  CALL HOTEL
"BOOK AIR"
  CALL AIR    ; this remark explains this selection
"EXIT SABREscript"
  EXIT
ENDMENU
```

Using the Developer

MySabre Scribe Developer is specifically designed for developing scripts. It contains unique features and functions that can assist you in development of scripts.

For information about the installation process refer to the *MySabre Scribe* Installation Guide.

Overview

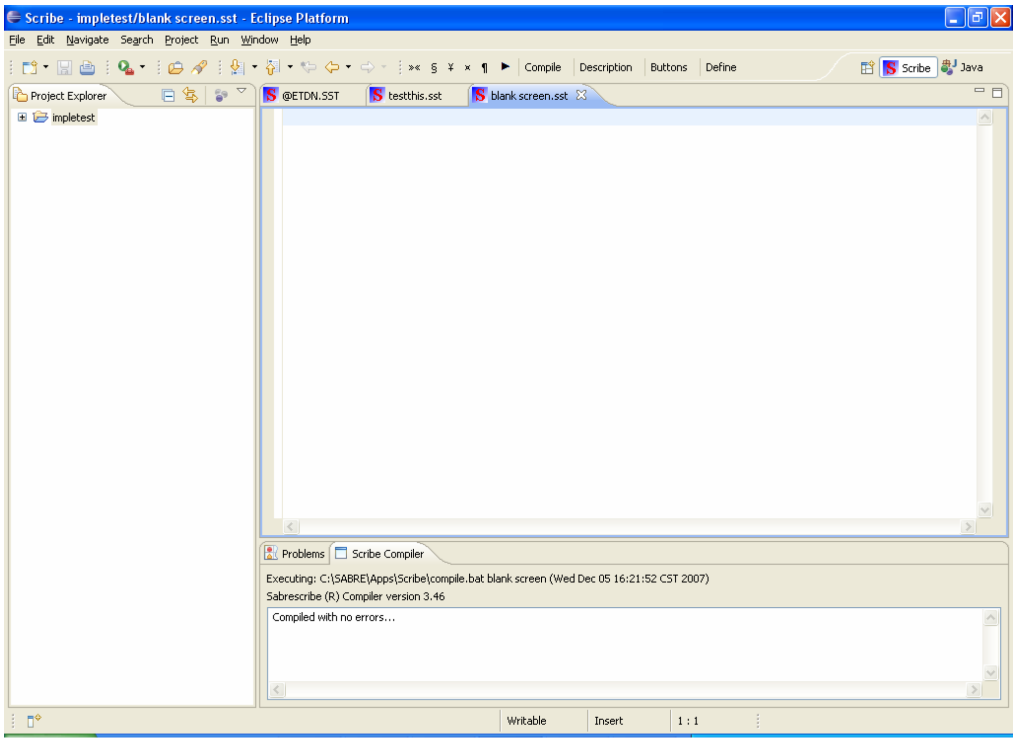
This section provides an overview of the *MySabre Scribe Developer* and its features. For more detailed information, refer to the online help available with the Editor.

Starting the Editor

To start the *MySabre Scribe Developer*, simply click the Eclipse icon, which is placed on your desktop after installation process, as shown below.

Editor

The *MySabre Scribe Developer* (accessed via the Eclipse platform) enables you to easily access features and functions. The Developer includes a Menu Toolbar, Special Character pallet, script-editing window, Project Explorer, and Compiler, as shown below:

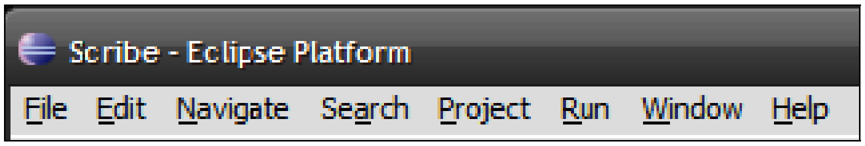

Scribe - impletest/blank screen.sst - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Compile Description Buttons Define
Scribe Java
Project Explorer
impletest
@ETDN.SST testthis.sst blank screen.sst
Problems Scribe Compiler
Writable Insert 1 : 1

```
Executing: C:\SABRE\Apps\Scribe\compile.bat blank screen (Wed Dec 05 16:21:52 CST 2007)
Sabrescribe (R) Compiler version 3.46
Compiled with no errors...
```

Menu Toolbar

The *Menu Toolbar* is located at the top of the Eclipse window and enables you to access most of the Developer's functions.

Note: The Menu Toolbar includes a variety of functions and features used by Eclipse which are not used by the *MySabre Scribe Developer*. This section will only address the menu functions that are applicable to the Developer.

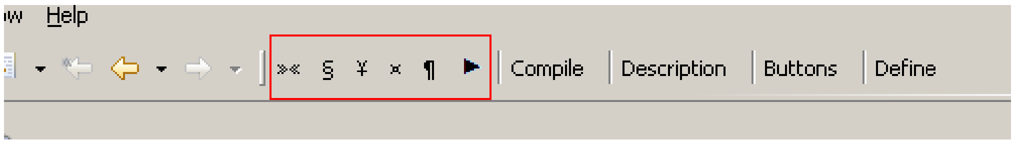

Scribe - Eclipse Platform
File Edit Navigate Search Project Run Window Help

- **File –** Create a new file, open it, save it, close it, and revert to last saved version.
- **Edit -** Cut, copy, and paste different parts of your script. Find a character string or replace a character string in your script.
- **Navigate –** "Go to line" option takes you directly to an error mentioned in the compiler.
- **Search -** Search for text strings in one or more files.
- **Run –** This menu is not applicable to *MySabre Scribe Developer*.
- **Window -** Change preferences, such as color choice and font.
- **Help –** Access information about Eclipse and Plug-in details. The *MySabre Scribe Developer* help system is not attached to this Help button. The Developer's help system is the "Auto Complete" feature (described later in this section).

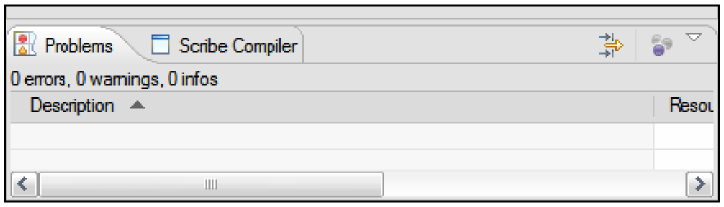
Editor Buttons

The *Developer* contains multiple buttons, which help you during the editing process, including the following:

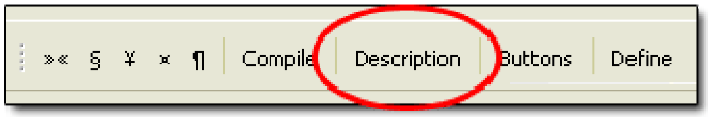
- **Special Character buttons** (**"»« ", "§", "¥", "¤" , "¶", "▶"** (right-pointed triangle for Menu Cascade) - Click them to insert appropriate symbols into the script you are editing.

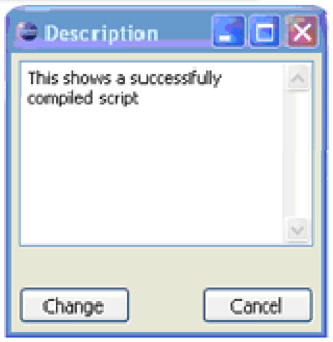


Compile button – Compiles the currently opened script and displays compilation output in the "Scribe Compiler" view located at the bottom of the Eclipse screen.



Description button – Brings up a window with a text field where a *MySabre Scribe* Script description may be defined. This description is later visible in *MySabre* “open script” window.

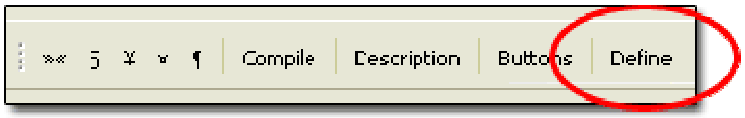


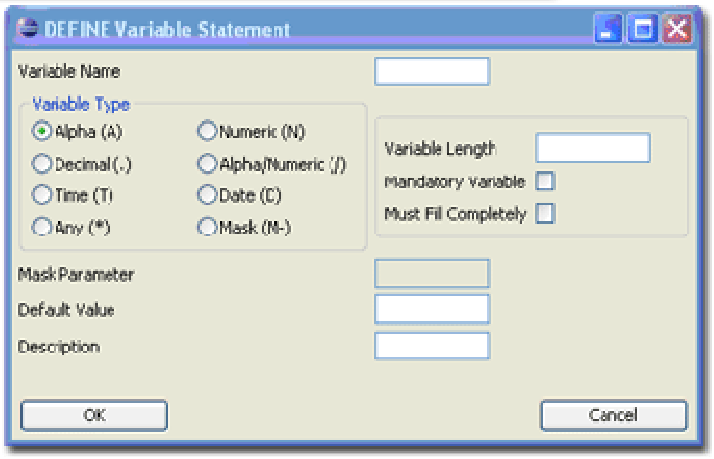


This shows a successfully compiled script

Note: Descriptions can now be edited at any time. In the previous version of *Sabre Scribe*, descriptions could only be set while using the “save as” option (without the possibility of editing an existing description).

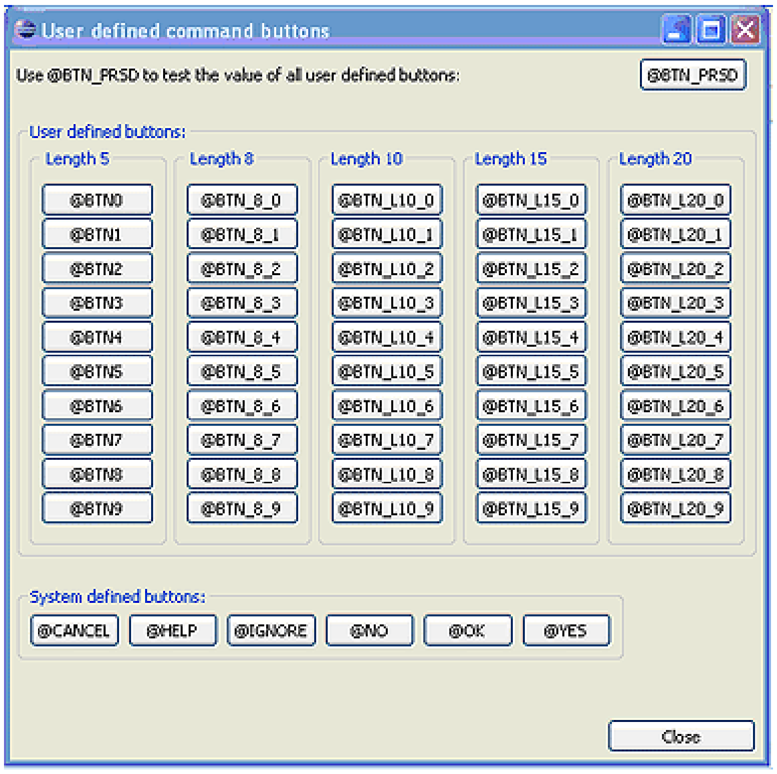
- **Define button –** Located at the top of the Eclipse screen, this button brings up a window which allows you to create a “define statement” with its properties, such as variable name, type, length, etc.





Variable Name	
Variable Type	[✓ Alpha (A)] Numeric (N) Decimal (.) Alpha/Numeric (/) Time (T) Date (C) Any (*) Mask (M-)
Variable Length	
Mandatory Variable	☐
Must Fill Completely	☐
Mask Parameter	
Default Value	
Description	

Buttons button – Brings up a window which allows you to create user defined variables without typing them. You only need to click on the appropriate button.


Use @BTN_PRSD to test the value of all user defined buttons:

Using Special Characters

MySabre Scribe Developer provides you with a pallet of Special Characters.

In order to enter special Sabre characters (»«§‡¤¶▶) into the currently edited script, click on the appropriate button on the toolbar. The "»«" characters are combined on one button. After clicking this button, both symbols will appear with the cursor between them.

Syntax Highlighting

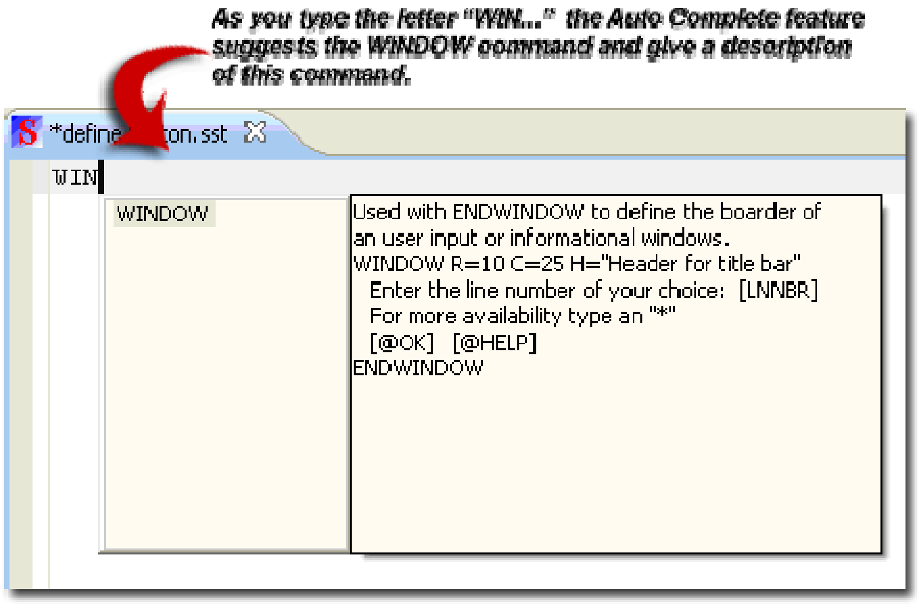
Highlights specific syntax elements while they are being written. When the syntax element is completed, it is colored and, in some cases, bolded.

Syntax Error Reporting

Automatically highlights / underlines syntax errors in a script. This feature can be turned on or off by selecting the Menu Toolbar option: **Window** >**Preferences** > **MySabre Scribe** >**Syntax Error Reporting.**

Auto Complete

Language structures such as `WHILE` or `IF-ENDIF` and Emulator commands such as `{ENTER}` or `{HOME}` can be auto completed. To display a list of Auto Completion candidates, press `CTRL+SPACE`. Input of further characters will narrow the proposals list.


As you type the letter "WIN..." the Auto Complete feature suggests the WINDOW command and give a description of this command.
*define...on.sst
WIN
WINDOW

```
Used with ENDWINDOW to define the boarder of
an user input or informational windows.
WINDOW R=10 C=25 H="Header for title bar"
  Enter the line number of your choice:  [LNNBR]
  For more availability type an "*"
  [@OK]  [@HELP]
ENDWINDOW
```

Useful Shortcut Keys

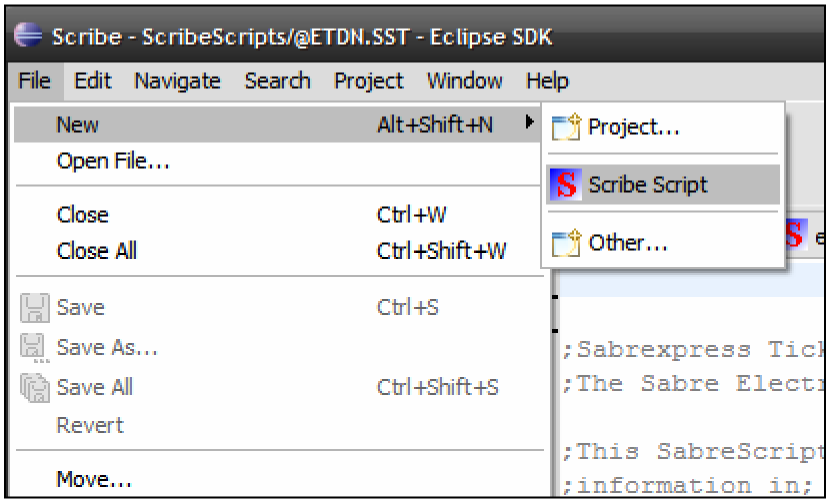
The following keys and key combinations can help you access functions quickly.

Key	Description
Ctrl+S	Saves currently opened script"
Ctrl+Shift+R	Opens scripts by typing its name or part of it
Ctrl+D	Deletes current line
Ctrl+C	Copies selected text
Ctrl+V	Pastes copied text
Ctrl+X	Cuts selected text
Ctrl + L	Allows you to type in the line number that the compiler mentions so that you can go directly to that error

All other shortcut bindings may be found inside Eclipse **(Help >Key Assists)**.

To create a script, make sure you have selected Scribe Perspective and that either your Project Explorer (containing scripts) or one of those scripts is selected.

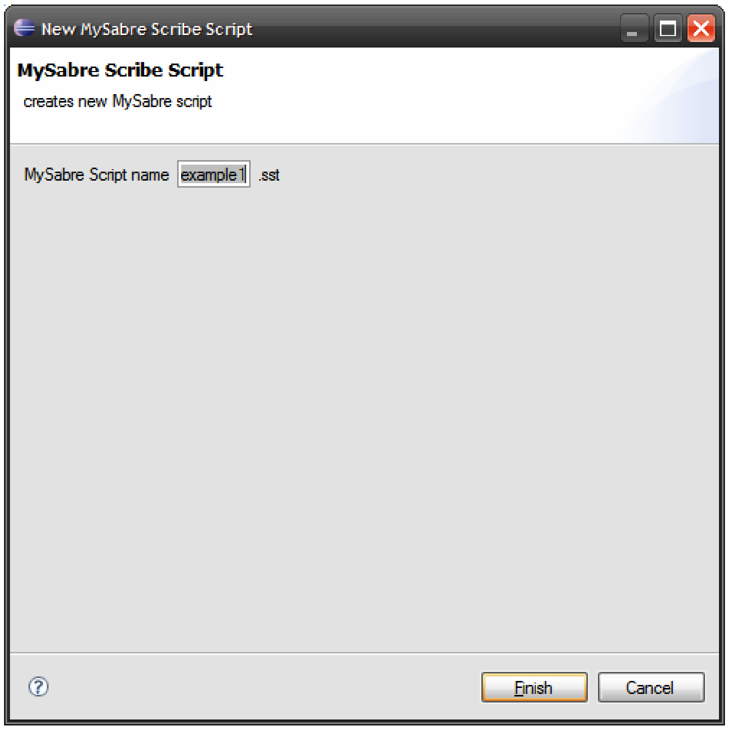
1. Select **File >New >Scribe Script**.



```
;Sabrexpress Tick
;The Sabre Electr

;This SabreScript
;information in;
```

The *MySabre Scribe Script* wizard displays.



MySabre Script name example1 .sst

2. Enter the name of the script. (Do not add an extension. The ".sst" extension is added automatically.)

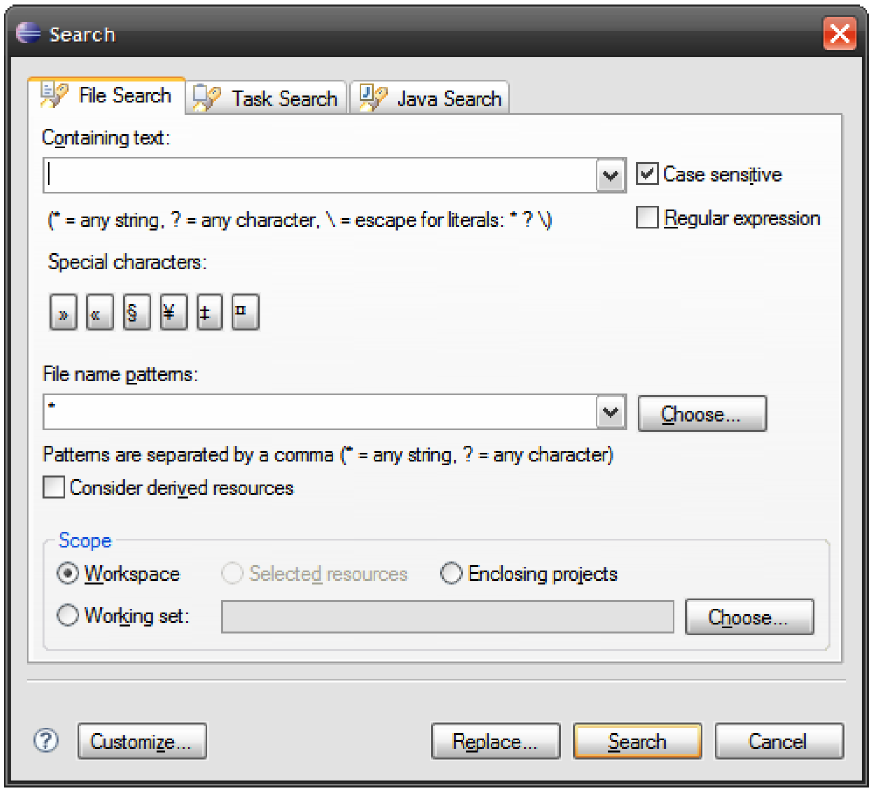
3. Click **Finish**.

MySabre Scribe Developer provides you with the ability to search for a text string (including Special Characters). The tool returns a list of files, indicating the place(s) in each file that the text string occurs.

Performing a search

To search for particular text in multiple scripts (which you see in the Project Explorer):

1. Select **Search >Search** or press **CTRL+H**.
   The Search window displays as shown below.



2. Enter the text you are looking for in the **Containing text** field.

Note: If you want to search for text with special Sabre character, press the corresponding button from **Special characters** section.

Advanced search options

There are several advanced options you can use to refine your search:

- **Case sensitive –** Only exact matches will be found, including case structure.

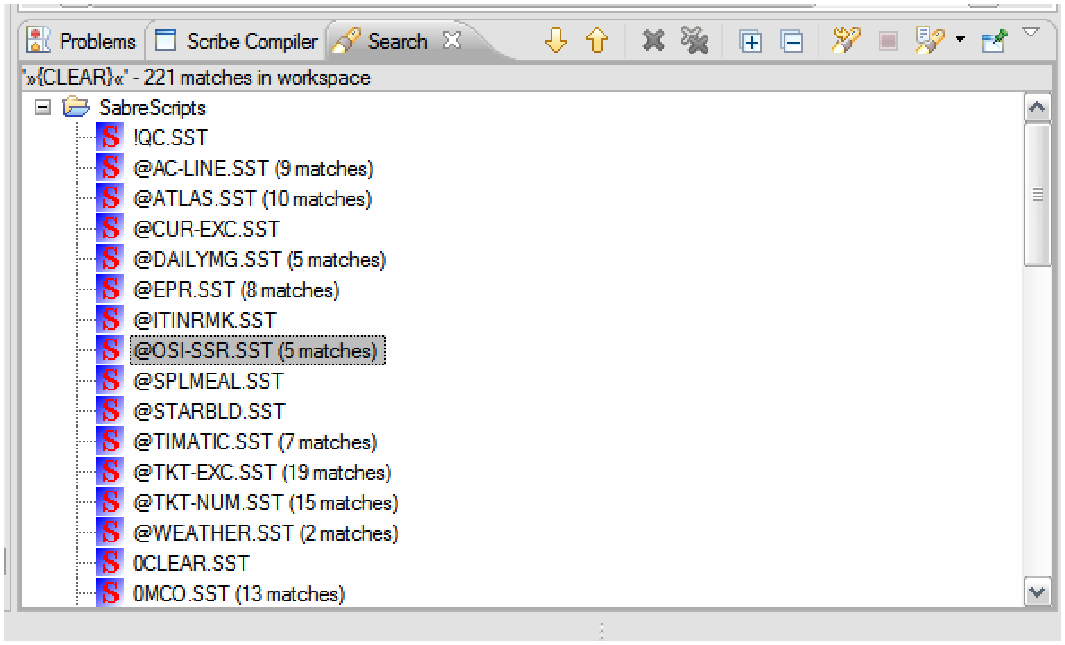
Note: If this option is not selected, the search is not case sensitive. For example, the following strings would be considered matches: "some_text", "some_text" ,"SOME_TEXT", and "somE_TExt"

- **Regular expressions -** Searches in regular expression mode.

- **File name patterns** enables you to limit the search area by defining file name pattern. For example, if you define the name pattern as **"@.sst"**, it means you want to search through all files that start with **'@'** and have an **".sst"** extension.
  - You can also define a file type (i.e. .sst) by picking it up from list after clicking **Choose** button.
  - **An asterisk (*)** means that you want to search through all files. This is the default pattern.

Displaying results

3. After defining your search parameters, press **Search.**
   Results display in the *Search View* at the bottom of the Eclipse screen, as shown below.



'»{CLEAR}«' - 221 matches in workspace

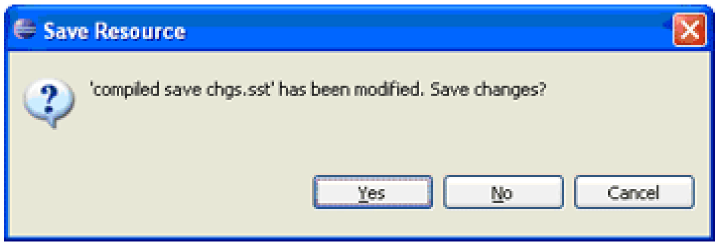
SabreScripts
- !QC.SST
- @AC-LINE.SST (9 matches)
- @ATLAS.SST (10 matches)
- @CUR-EXC.SST
- @DAILYMG.SST (5 matches)
- @EPR.SST (8 matches)
- @ITINRMK.SST
- @OSI-SSR.SST (5 matches)
- @SPLMEAL.SST
- @STARBLD.SST
- @TIMATIC.SST (7 matches)
- @TKT-EXC.SST (19 matches)
- @TKT-NUM.SST (15 matches)
- @WEATHER.SST (2 matches)
- 0CLEAR.SST
- 0MCO.SST (13 matches)

After you create your script using the Developer, you will need to compile it.

Using the Compiler

To compile a currently opened script:

1. Click the **Compile** button.
2. A window prompts you to save changes. Click **Yes.**

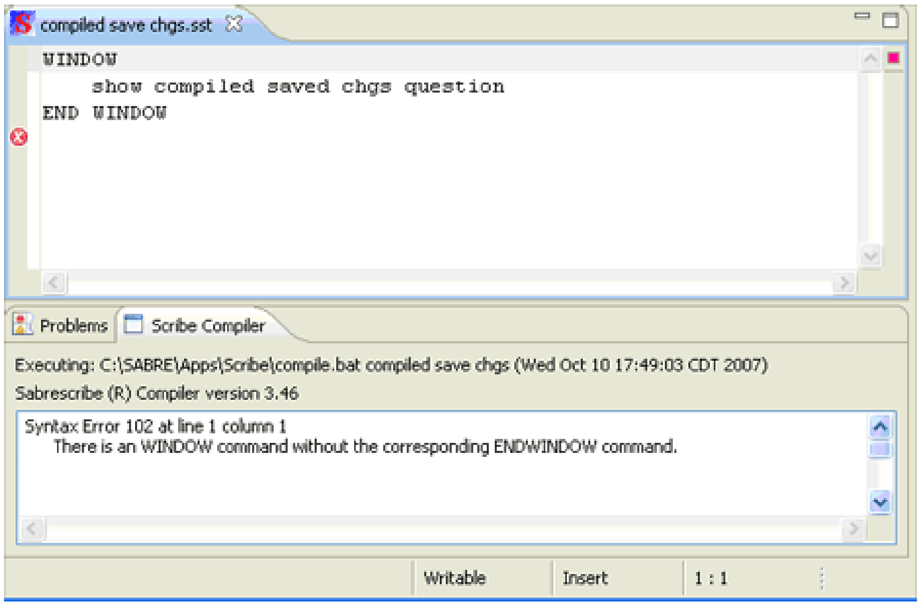


'compiled save chgs.sst' has been modified. Save changes?

3. If errors are found during the compiling process, correct the errors in your script and recompile it.

Viewing Compiler Errors

When you compile your script, the compiler edits commands for accuracy. If the compiler finds something it does not understand, the problem is displayed in the **Problems** window of the **Scribe Compiler** at the bottom of the Eclipse screen.



```
WINDOW
    show compiled saved chgs question
END WINDOW
```

```
Executing: C:\SABRE\Apps\Scribe\compile.bat compiled save chgs (Wed Oct 10 17:49:03 CDT 2007)
Sabrescribe (R) Compiler version 3.46
Syntax Error 102 at line 1 column 1
    There is an WINDOW command without the corresponding ENDWINDOW command.
```

Using Loops

A loop is used to group instructions that you want to execute in a repetitive manner. Loops allow you to produce efficient, readable scripts.

MySabre Scribe software enables you to use two types of loops: REPEAT/UNTIL and WHILE/ENDWHILE. These loops operate differently and have unique reasons for their use.

REPEAT/UNTIL

Use a **REPEAT/UNTIL** when you have a set of instructions you want to execute at least once before evaluating conditions (an expression) which determine whether you want to execute the instructions again.

Example:

```
.
.
.
REPEAT
    WINDOW
    R=1 C=13 S=0
    DO YOU WISH TO VIEW MORE AVAILABILITY? [MORE.Y/N]
    ENDWINDOW

    IF [MORE.Y/N] = "Y" THEN
           »{HOME}1*{ENTER}«
    ENDIF

   UNTIL [MORE.Y/N] = "N"
   .
   .
   .
```

This example picks up after a user has requested availability. A window displays asking if the user would like to request more availability. If the user answers “Y”, the *Sabre* system command is executed to display more availability and the window reappears prompting the user again. If the user answers “N”, the loop is terminated.

Notice the loop makes no evaluation when it starts execution. The evaluation is made at the end of the loop.

WHILE/END WHILE

Use a **WHILE/ENDWHILE** when you have a set of instructions you only want to execute after you evaluate conditions (an expression) which determines whether the instructions should be executed.

Example:

```
.
.
.
IF [MYOPT1] = "Y" THEN
        [MORE.Y/N] = "N"
ELSE
[MORE.Y/N] = "Y"
ENDIF
.
.
.
WHILE [MORE.Y/N] = "Y"
        WINDOW
        R=1 C=13 S=0
        DO YOU WISH TO VIEW MORE AVAILABILITY?[MORE.Y/N]
        ENDWINDOW

        IF [MORE.Y/N] = "Y" THEN
              »{HOME}1*{ENTER}«
        ENDIF

        ENDWHILE
        .
        .
        .
```

This example picks up after users request availability. Prior to starting to execute the loop, users specify whether they want to be prompted to request more availability during the script. When the loop first starts to execute, it evaluates the options. If the users specified that they would like to be prompted for more availability, the loop executes, displaying the window asking if more availability is desired. If the users specified that they did not want to be prompted, the control of the script jumps to the statement after the ENDWHILE command.

A routine is a group of instructions that you want to isolate from the body of your script. This is done when you want to execute a routine several times during the execution of a script or when you want to provide support for more than one action from a window.

MySabre Scribe software provides several types of routines:

- WHENEVER/ENDWHENEVER
- SUBROUTINE/ENDSUBROUTINE
- GOTO
- CHAIN
- SPAWN

WHENEVER/ ENDWHEN-EVER

Use **WHENEVER/ENDWHENEVER** if you want to execute a set of instructions whenever a specific condition becomes true. This condition could be a user action or a program condition.

Example:

```
WHENEVER [@HELP]
        WINDOW
        R=13 C=5 S=1 H="Car Help System"
        This text can be anything you wish to help the
        agent understand the functionality of the "Car
        Availability" Window. You can create your own
        HELP system with anything you wish!
         (Press any key to return to the previous Window)
        ENDWINDOW
ENDWHENEVER

WINDOW
R=1 C=5 S=0 H="Car Availability"
        RENTAL CITY: [RENTAL_CITY]   CAR VENDOR: [CAR_VENDOR]
        PICKUP DATE: [PKUP_DATE]     PICKUP TIME: [PKUP_TIME]
DROP-OFF DATE: [DROP_DATE]    DROP-OFF TIME: [DROP_TIME]

                [@HELP]
ENDWINDOW
```

In this example, WHENEVER/ENDWHENEVER is used to support more than one task. The primary task is obtaining car availability information. The secondary task is understanding the fields on the window.

If the user presses the @HELP button on the Car Availability window, the help window displays. When the user finishes reading the help information and presses a key, the Car Availability window reappears. The user can then complete the Car Availability window or request the help window again.

SUBROUTINE/ENDSUBROUTINE

Use **SUBROUTINE/ENDSUBROUTINE** if you have a group of instructions that your script needs to execute several times during the overall execution of the script. This enables you to reduce the total code you create and test. This type of routine is invoked with a CALL and SUBROUTINE commands. After executing the subroutine, the script executes the instruction immediately following the CALL command.

Example:

```
.
.
.
»{CLEAR}«
»Y/SAB/HOT{ENTER}«
CALL MOVE
WINDOW
R=1 C=80 S=0
Have a nice day.
ENDWINDOW
.
.
.
»{CLEAR}«
»Y/INT/HOT/32{ENTER}«
CALL MOVE
.
.
.
SUBROUTINE MOVE:
MENU
R=1 C=80 S=0 B=1
    "Move &Down"
        »MD{ENTER}«
    "Move &Up"
        »MU{ENTER}«
    "Move &Top"
        »MT{ENTER}«
    "Move &Bottom"
        »MB{ENTER}«
    "Target &Word"
        WINDOW
        R=1 S=0 H="Select Target M3"
        Enter the target word for the Move Down to target: [TARGET]
    ENDWINDOW
    DEFAULT [TARGET] = [TARGET]
    »MD/[TARGET]{ENTER}«
–
    "Return to &Menu"
        RETURN
–
    "E&xit script"
        IF [SPLIT] = "Y" THEN
        »{SPLIT}«
    ENDIF
    EXIT
ENDMENU
ENDSUBROUTINE
.
.
.
```

Note: When the subroutine MOVE ends, the script executes the line immediately following the CALL command. In this example, the “Have a nice day.” window appears after the first call to MOVE.

The software provides a RETURN command (shown in the previous example) that allows you to exit a subroutine and execute the instruction immediately following the CALL command that invoked the subroutine. This is useful when the processing of a subroutine indicates that no further actions are necessary.

You should never leave a subroutine with a GOTO command. Leaving a subroutine with a GOTO produces an poorly formed script. It also uses up memory. If you leave a subroutine with a GOTO command, the subroutine remains in memory, waiting for completion (ENDSUBROUTINE or RETURN).

GOTO

Use **GOTO** if you have a group of instructions you want your script to jump over. When you use this command, the script looks for a label. It ignores any instructions it finds along the way.

Example:

```
WINDOW
R=1 C=5 S=1 H="Air Availability"
ORIGIN: [ORIGIN_CITY]  DESTINATION: [DESTIN_CITY]
DEPART: [DEPART_DATE]         TIME: [DEPART_TIME]
ENDWINDOW

IF [DESTIN_CITY] = "BOS" THEN
       GOTO BOSTON
ELSEIF( ([DESTIN_CITY] = "ORD") OR ([DESTIN_CITY] = "CHI")) THEN
       GOTO CHICAGO
ENDIF

BOSTON:
       WINDOW
       Be sure to explain ABC Corporation’s Car Pool Policy for
       Boston.
              (Press any key to execute airline availability)
       ENDWINDOW

       GOTO CONTINUE

CHICAGO:
       WINDOW
       Be sure to explain ABC Corporation’s Car Pool Policy for
       Chicago.
              (Press any key to execute airline availability)
       ENDWINDOW

CONTINUE:
   »1[DEPART_DATE][ORIGIN_CITY][DESTIN_CITY][DEPART_TIME]{ENTER}«

EXIT
```

Note: The label name can be from one to 79 alphanumeric characters. It must be followed by a colon. The first character of a label name must not be numeric. A label name must not be a command.
A label can be placed anywhere in a script.
Unlike a the SUBROUTINE/ENDSUBROUTINE command, a label has no ending logic; therefore, the script does not return to the instructions immediately following the GOTO command. It continues in a normal top-to-bottom flow from the label to the end of the script.

CHAIN

Use **CHAIN** if you have a series of lengthy routines or another script that your script needs to execute *after* it stops. You can isolate the lengthy routines in separate scripts. Your script can then call the separate scripts whenever you invoke them using the CHAIN command. When the separate script is invoked, the original script execution stops. Control moves to the new script. When the new script ends, the processing is complete. You can also use external variable with the CHAIN command.

Example:

```
; MAIN SCRIPT
  .
  .
  .
  WINDOW
  R=1 C=80 S=0
  Origin City: [ORIGIN.CTY]     Destination City: [DESTIN]
  Departure Date: [DEPART]    Departure Time: [DEPART.TIME]
  ENDWINDOW

  IF (([DESTIN.] >= "AAA")  AND  ([DESTIN] < ="AZZ"))  THEN
       CHAIN "CITYA"
  ELSEIF (([DESTIN] >= "BAA")  AND ([DESTIN] < ="BZZ"))  THEN
       CHAIN "CITYB"
  ELSEIF (([DESTIN] >= "CAA")  AND ([DESTIN] < ="CZZ"))  THEN
       CHAIN "CITYC"
  ENDIF
  WINDOW
  R=1 C=80 S=0
  This script currently only handles city codes from A - C.
  Have a nice day.
  ENDWINDOW
  EXIT
```

In this script, MAIN, a window displays prompting the user to enter travel information that includes the destination city/airport codes. If the user enters the airport code “BOS”, the software invokes the CITYB script (see below) and terminates MAIN. The last window in MAIN would only appear if the user provided a city code not handled by the IF/THEN/ELSEIF/ENDIF statements, for example, DEN. After displaying the window, the script would end.

```
; CITYB SCRIPT
  DEFINE [DESTIN]

EXTERNAL[DESTIN]

DEFINE [NAME=ALPHA:20:OPT:::]


IF [DESTIN] = "BAO" THEN [NAME] = "BAN MAK KHAENG, THAILAND"

ELSEIF [DESTIN] = ‘BMV’ THEN [NAME] = "BAN ME THOUT, VIETNAM"
ELSEIF [DESTIN] = ‘BDY’ THEN [NAME] = "BANDON, OREGON"

ELSEIF [DESTIN] = ‘BOS’ THEN [NAME] = "BOSTON"
  .
  .
  .
  EXIT
```

Note: When CITYB ends, control does not return to MAIN since MAIN was terminated by use of the CHAIN command.

SPAWN

Use **SPAWN** if you have a series of lengthy routines or other scripts that your script needs to execute *during* its processing. You can isolate the lengthy routines in separate scripts. Your script can then call the separate scripts whenever you invoke them using the SPAWN command. When the separate script is invoked, the original script execution pauses. Control moves to the new script. When the new script ends, control moves back to the original script.

Example:

```
; MAIN SCRIPT
.
.
.
»NAN,CCARS-{ENTER}«
IF [CID1] THEN
     [CID] = [CID1]
     [CIN] = [CIN1]
     [CIDN] = [CIDN1]
     SPAWN "PT_CID"
ENDIF
WINDOW
R=1 C=80 S=0
Have a nice day.
ENDWINDOW
EXIT
```

This script, MAIN, invokes a second script, PT_CID, based on a condition. When PT_CID is invoked, MAIN halts until PT_CID completes. When PT_CID completes, MAIN resumes operation at the next instruction. In this example, it displays the window.

```
; Decode Car Vendor SCRIPT
;
;Hidden File
DEFINE [CID=/:2:O:::Car Company]            EXTERNAL [CID]
DEFINE [CIN=*:15:O:::Car ID Nbr]            EXTERNAL [CIN]
DEFINE [CIDN=*:20:O:::Car ID Desc]          EXTERNAL [CIDN]
DEFINE [DUMMY=*:1:O:::Dummy Var]
DEFINE [S=*:6:O:::SABRE]
[S] = "SABRE:"
»NAN,C«
IF [CID] = "AC" THEN »ACE           [CIN]«
ELSEIF [CID] = "AD" THEN »ADVANTAGE     [CIN]«
ELSEIF [CID] = "AI" THEN »AMERICAN INTL [CIN]«
ELSEIF [CID] = "AL" THEN »ALAMO         [CIN]«
ELSEIF [CID] = "CC" THEN »TOWN AND COUN [CIN]«
ELSEIF [CID] = "ED" THEN »EURODOLLAR    [CIN]«
ENDIF
.
.
EXIT
```

Note: When PT_CID ends, control returns to MAIN.
You can use external variables when spawning scripts to avoid needing to request data again. If you use external variables when using a SPAWN command, define the external variable in both the main and spawned scripts since the variable will be active during the execution of both.

MySabre Scribe software allows you to read information into a script. The software puts each character of the information into a variable until it

- Finds an invalid character type for the variable
- Reads the maximum number of characters for the variable
- Finds a character defined as a terminating expression

You can also specify more than one variable to read data into. The data is read starting from the row and column position you specify. If a row and column position is not specified, the software reads the last position of the data source.

You can use READ in conjunction with OPEN and CLOSE commands to better manipulate the files from which you read. For more information, see “Opening and Closing Files.”

Terminating a READ Command

You can terminate a READ command by

- Specifying the length of the variable you read
- Anticipating a change in the type of data being read
- Using a terminating character

When you define a variable for a READ command, you specify the length of the variable. The read command will only read the number of characters specified in the variable.

Example:

```
DEFINE [NBR=N:4]
READ FILE="EMULATOR:" R=2 C=13 [NBR]
```

In the previous example, if the text on the *Sabre* desktop reservations system, row 2 was "` 14A  5TEST 123456`", the script would read "1234" into the variable NBR because the variable length is 4. Note that there are two spaces between `14A` and `5TEST`.

In the previous example, if the text on the *Sabre* desktop reservations system, row 2 was "` 14A  5TEST 123ABC`", the script would read "123" into the variable NBR because the variable type changed at the 16th position on the row.

Example:

```
DEFINE [NBR=N:4]
DEFINE [NBR2=N:4]
READ FILE="EMULATOR:" R=2 C=13 [NBR], "3", [NBR2]
```

In the previous example, if the text on the *Sabre* desktop reservations system, row 2 was "`14A  5TEST 123456`", the script would read 12 into the variable NBR because the terminating character specified was a 3. If you read multiple variables and use a terminating character, the script skips over all the characters matching the terminating expression and continues reading the next characters. In this example, the script reads 456 into the variable NBR2 because the 3 was skipped.

Example:

```
DEFINE [CODE=*:40]
READ FILE="EMULATOR:" R=2 C=1 [CODE], "*"
```

In the previous example, if the text on the desktop, row 2 was "`SMITH/FRANK*DFW`", the script would read SMITH/FRANK into CODE because a display character was specified as the terminating character.

Reading Data from the Sabre Desktop Reservations Software

Use **FILE="EMULATOR:"** when you want to read data that is already visible to the user.

Note: You must first determine the column and row position where you want to start reading the data. Keep the following in mind:

- Make sure the position is consistent—the same position, regardless of what the script user was doing before the read is executed.
- The first row of the active partition counts as the first row. Example: If the full screen is the active partition, specify the second line of the full screen with R=2. Do not specify the first line of the display (R=1) because the Sabre system response is in line 1.

Note: Include a DEFINE command for the data you want to read. The variable length must match the number of characters you want to read. Remember to include spaces as part of the length. If the length of the data varies, use the maximum length.
Include a READ FILE="EMULATOR:" command with the row, column, and variable that you identified. Also include the terminating character, if necessary.

Example:

```
DEFINE [LAST_NAME=A:25]
DEFINE [FIRST_NAME=A:25]

»{CLEAR_ALL}*N{ENTER}«
READ FILE="EMULATOR:" R=2 C=4 [LAST_NAME], [FIRST_NAME]
```

In the previous example, the variables defined for last and first names are ALPHA. Since the characters immediately following each are non-ALPHA, providing the terminating character is not necessary. The READ command reads the data until it finds the character type that does not match the variable type.

We know the data is in R=2 C=4 because we positioned it there with the {CLEAR_ALL} command.

If you know the row position, but you are uncertain about the column position, you can use a loop to find the correct column position. Similarly, you can look for a row position if you know the column position. The following example illustrates a situation where you know the row position.

Example:

```
DEFINE[LOOK=*]
DEFINE [COLUMN=N:2]
DEFINE [NAMEPOS=N]
[NAMEPOS]=0
[COLUMN]=5
   »{CLEAR_ALL}*N{ENTER}«
REPEAT
   REPEAT
         READ F="EMULATOR:" R=2 C=[COLUMN] [LOOK]
         [COLUMN]=[COLUMN]+1
   UNTIL [LOOK]="/"
   [NAMEPOS]=[NAMEPOS]+1
UNTIL [NAMEPOS]=2
```

The previous example demonstrates a technique to locate the second passenger's first name. Since that location varies with each PNR, we read character by character until the second "\" is encountered.

Note: ***The EMUFIND device (described in the Scripting Reference Guide) allows parsing of the emulator screen using strings, regular expressions, or number of characters to skip*.**

Reading Data from the Sabre System

Use **FILE="SABRE:"** when you want to read data from the *Sabre* system without displaying it to the user.

Note: Include a DEFINE command for the data you want to read. The variable length must match the number of characters you want to read from the *Sabre* system. Remember to include spaces as part of the length. If the length of the data varies, use the maximum length.
Issue a WRITE FILE="SABRE:" command to make a *Sabre* system entry immediately prior to the READ command. You should verify the entry and response prior to developing your script to ensure your results are accurate.

Example:

```
DEFINE [END=A:6::::"NO NAM"]

AGAIN:
WRITE FILE="SABRE:" "DIT"
READ FILE="SABRE:" [END]
IF [END]="END OF"  THEN
   EXIT
   ELSE GOTO AGAIN
   ENDIF
```

In this example, DIT is written to each PNR on the queue. When the script reaches the end of the queue, it exits. The READ command in this script verifies that the response begins with "END OF" each time the DIT is written to a PNR. This is consistent with the first six characters of the *Sabre* system response, "END OF DISPLAY FOR REQUESTED DATA".

Reading Data from an ASCII File

Use **FILE=expression** when you want to read data from an ASCII file. The expression must evaluate to the fully qualified name of the file.

The file could be a file in a standard delimited format or a spreadsheet that separates each piece of data with a comma, or an editor that separates each line with a carriage return. It could also be in a format that is nonstandard.

Example:

```
DEFINE [LAST=A:27]
DEFINE [FIRST=A:13]
DEFINE [SALES=DEC:7]
DEFINE [TOTAL=DEC:8]
DEFINE [NUM_IN_GROUP=NUM:1]

[TOTAL]=0
[NUM_IN_GROUP]=0
REPEAT
   READ F="C:\WINDOWS\TESTDATA.TXT"
        R=[NUM_IN_GROUP] C=1
        [LAST],  ",", [FIRST],  "," [SALES]
   [TOTAL] = [TOTAL] + [SALES]

   [NUM_IN_GROUP]=[NUM_IN_GROUP]+1
UNTIL [NUM_IN_GROUP] > 5
```

In this example, the READ statement obtains the first and last name, along with associated sales and totals of all the sales.

Reading Data from the Windows Clipboard

Use **FILE="CLIPBOARD:"** when you want to read data from the Windows clipboard.

Note: You should use this option when you have recently written data to the clipboard and are sure of its contents.
You must first determine the column and row position where you want to start reading the data. Make sure the position is consistent—the same position, regardless of what the script user was doing before the read is executed.
Include a DEFINE command for the data you want to read. The variable length must match the number of characters you want to read from the clipboard. Remember to include spaces as part of the length. If the length of the data varies, use the maximum length.
Include a READ FILE="CLIPBOARD:" command with the row, column, and variable that you identified. Also include the terminating character, if necessary.

Example:

```
   DEFINE [TEXT=*:50]
   DEFINE [COUNT=N:2]

   [COUNT]=0
   WHILE [COUNT]<99
      READ F="CLIPBOARD:" [TEXTA]
            »NAA [TEXT] {ENTER}«
       [COUNT]=[COUNT]+1

   ENDWHILE
      »NAA TOTAL LINES ADDED IS [COUNT] {ENTER}«
```

In this example, the READ statement obtains lines of text from the clipboard and inserts them into a *Stars* customer profile.

MySabre Scribe software allows you to write information to various destinations, such as the *Sabre* system, the *Sabre* desktop reservations software, or an ASCII file.

The data that you write comes from a variable or a character string. You place this data in your WRITE command in the sequence you want it written to the destination. You can place multiple variables or strings in one WRITE command.

If more than one variable or string is specified, each one must be separated by a comma.

The following sections describe how to write data to different sources.

You can use WRITE in conjunction with OPEN and CLOSE commands to better manipulate the files to which you write. For more information, see “Opening and Closing Files.”

Writing Data to the Sabre System

Use **FILE="SABRE:"** when you want to write data to the *Sabre* system without displaying it on the desktop reservations software.

You must use this command if you want to READ data directly from the *Sabre* system. For more information, see “Reading Data from the Sabre System.”

Example:

```
DEFINE [VAR=ANY:35:M]
[VAR]=”TEST”
WRITE FILE="SABRE:" "MD/" + [VAR]
```

In this example, the *Sabre* system command, MD/TEST{ENTER}, is written to the system and the processing occurs, but the user does not see the command.

Writing Data to an ASCII File

Use **FILE=”filepath”** when you want to write data to an ASCII file. The expression must evaluate to the fully-qualified name of the file.

Note: If it is a comma delimited format, use the CSV technique.

Example:

```
DEFINE [LAST=A:27]
DEFINE [FIRST=A:13]
DEFINE [SALES=DEC:7]
DEFINE [NUM_IN_GROUP=NUM:1]
DEFINE [PATH=ANY:48;MAND:::PATH OF DATA FILE]

[NUM_IN_GROUP] = 0
[PATH]=”H:\\PUBLIC1\\WINAPPS\\SCRIBE\\COMPILED\\SALESTOT.DAT”

REPEAT
   WINDOW
   First  [FIRST]
   Last   [LAST]
   Sales  [SALES]
   ENDWINDOW
```

```
      WRITE FILE=[PATH] [FIRST], [LAST], [SALES]
      [NUM_IN_GROUP]=[NUM_IN_GROUP]+1
   UNTIL [NUM_IN_GROUP]>5
```

In this example, the WRITE command places the last name, first name, and total sales for each salesperson into a file, SALESTOT.DAT. These can be retrieved later for use. Notice that the path is defined and the file is written to the path.

Writing Data to the Windows Clipboard

Use **FILE="CLIPBOARD:"** when you want to write data to the Windows clipboard. You must use this command if you want to read data from the Windows clipboard. For more information, see "Reading Data from the Windows Clipboard."

Writing Data to the Printer

Use **FILE="PRINTER:"** when you want to write data to a printer defined in the Printers icon in the Control Panel.

Note: You may specify row and column positions, but you may only move forward. You cannot move backwards. For example, you could not print on row 5 and then print on row 2. You must first print on row 2, then on row 5. To start a new page, use the terminating character "\f".
Because Windows spools your print jobs until it finishes, you must close the printer in order to retrieve the last page. (Example: CLOSE "PRINTER:")

Example:

```
   DEFINE [LAST=A:27]
   DEFINE [FIRST=A:13]
   DEFINE [SALES=DEC:7]
   DEFINE [NUM_IN_GROUP+NUM:1]
   [NUM_IN_GROUP] = 0
   REPEAT
      WINDOW
      First  [FIRST]
      Last   [LAST}
      Sales  [SALES]
      ENDWINDOW
      WRITE F="PRINTER:" [LAST], ",", [FIRST], ",", [SALES]
      [NUM_IN_GROUP]=[NUM_IN_GROUP]+1
   UNTIL [NUM_IN_GROUP]>5
   CLOSE "PRINTER:"
```

In this example, the salesperson's name and total sales are written to the printer along with the total sales.

Opening and Closing Files

MySabre Scribe software automatically opens and closes files when you use WRITE and READ commands. The software also provides the separate OPEN and CLOSE commands to allow you more flexibility in manipulating ASCII files.

Use the **OPEN** command when you want to leave a file open until the script encounters a CLOSE command. For example, you may want to read data from a file, manipulate that data, then continue reading data at the point you originally stopped.

Use the **CLOSE** command when you are finished using the file.

Techniques for Developing Scripts

Understanding the Process

This section covers the use of several components of *MySabre Scribe* software in building your new scripts. It also includes helpful hints for making your scripts more efficient and useful in day-to-day operations of your agency.

Steps for Creating Successful Scripts

1. Identify a problem area or request.

   Check with your office’s Customer Service Department (or the person who responds to complaints) to find the top problem areas in the “front-room” reservations operation. Look for cases such as overlooked special meals or quality control problems with itineraries before ticketing that can cause customer problems. Concentrate on frequently occurring problems or high priority agent requests for automation of routine transactions.

2. Determine a suitable solution.

   Develop several approaches to each problem and review your solutions with several people close to the process or problem area. Make sure that you understand the problem thoroughly and that your proposed solution meets your objectives. Verify that you have gathered information about the functions to be included in the script.

3. Create a flow chart of the scripts.

   Create a detailed flow chart of your script program illustrating the step-by-step approach the script will follow. Use it as a tool to present your design for the functions in your script to other people in the office. This feedback can help you reshape your design or add elements that were overlooked initially. The flow chart should outline the precise order, identifying

   – *Sabre* system formats to be executed and the area of the screen to which the responses will be directed

   – Choices to appear on the menus and the commands to be executed when each is selected

   – Information the agent will be prompted to input or the commands to interpret the *Sabre* system responses instead of prompting the agent for information

4. Create the script.

   Create the script, following the flow chart you outlined. Make full use of the commands available to complete the tasks needed for your agency. Add comments to your scripts to ensure any changes or later additions are easy to complete.

5. Test the script.

As you create your script, test it periodically. Make sure all the windows, menus, Sabre system commands, and variables perform the way you designed them.

- Go through your script (every possible path) and enter things you would and would not expect. If you prompt the user to enter a “Y” or “N” into a variable, enter an “X”. Does your program logic still work or does the script “break”?
- Test various combinations of entries, especially those involving optional fields. Whether the optional fields are used or not, does the script execute as it should?
- Does your script execute the correct *Sabre* system commands in the correct locations of the screen?
- Do the menus or windows obstruct the view of *Sabre* system responses? If so, shorten or reposition them as needed.
- Have agents in your office test your script after you have completed your own testing. Select two or three people in one area or department who will be using the finished script. Train them on the use of the script. Ask them to use it for a few days and report any discrepancies they may find.

6. Create supporting documentation.

   Produce a high-level overview describing the script and its primary function. This is used as a quick reference for each script in your office. Also create a detailed document to be given to the agents using the script. Documentation provides agents with the instructions or reminders needed to effectively use your script. Remember that supporting documentation never eliminates the need to use comments in your scripts to document the organization and flow.

7. Conduct training.

   A well-trained user can take full advantage of your new script. Announce the new script with a brief outline and encourage the agents to try it and ask questions. Explaining the purpose and time-saving features of the script prompts the agents to access and use new scripts.

Building Menus

Be consistent in the placement of your menus on the screen. This allows the agents to become familiar with your menus and look for them in the same place in each script. For example, if you position menus on row 1, column 80 of the screen, they are displayed on the far right side of the screen. This limits the amount of *Sabre* system display space covered by the menu display. Keep menu choices as short as possible, without sacrificing clarity. If necessary, abbreviate menu choices as long as the abbreviations are understandable. Shorter menu choices keep your menus as narrow as possible.

Limit the number of items listed in your menus. A long menu makes it awkward for agents to search for a selection. The new cascading menu feature enables you to keep each menu as short and simple as possible. You can group similar options in a secondary cascading menu. When a cascading menu option is selected, the primary menu remains inactive in the background while the secondary menu is displayed in the foreground. The agent can press ESC to exit the secondary menus until the primary menu is displayed. If ESC is pressed while in the primary menu, a message asks “Do you wish to terminate the sabreScript?”

There are other ways to create effective menus:

- Standardize the text used in your menu options. If you have a menu item which reads “SCROLL DOWN” in one menu, use the same text in other menus to avoid confusion.
- Place the most frequently selected options as the first items in your menu. Placing menu options in expected order of selection prevents a time consuming search through the menu in most cases.
- Use the menu item “Exit sabreScript” as the last item in all of your menus.
- Standardize your usage of accelerators in menus. For example, always use the accelerator “X” for exit, “D” for scroll or move down, and “U” for scroll or move up.
- Use dividing lines to emphasize groupings of selections in a menu.

Using Windows in Your Scripts

Develop a consistent style in the placement of windows as well as menus. The consistency helps agents become familiar with your script style.

Experiment with the length and type of responses *Sabre* system returns to your screen. This testing allows you to select the window position that displays the maximum information for the agent.

Emphasize mandatory variables in windows, so the agent understands that an entry is required for that input field. For example, use uppercase letters for the descriptive text of the input field or place a symbol, such as an asterisk “*”, in front of mandatory fields.

Use the same descriptive text each time you display a variable input field in your script. For example, using “ORIGIN” in one window and “BOARDING POINT” in another window can make the agent unsure that both variables are the same.

Avoid the use of the SCREEN command. Screens can obstruct displays from the *Sabre* system because they can only be sized for a full or split screen. Windows can be sized small enough and can be positioned to avoid covering *Sabre* system data.

Adding Help Information to Your Scripts

Use the new system variable @HELP to add help windows to your scripts. Help windows can contain reminders about the purpose of the input fields or questions in a window.

Help windows can be simple instructions without elaborate formats or content. They can reduce the time needed to complete a script and the awkward fumbling caused by an unexpected question.

See the WHENEVER/ENDWHENEVER command listings for more information about adding help to windows in your scripts.

You can also add a cascading menu option to display helpful information about the selections in your menus.

Both of these methods can assist your agents in completing tasks in your office and in explaining options available to your clients.

Preparing the Sabre Desktop Reservations Software

Remember that a script can be executed at any time, without verifying the current state of the *Sabre* desktop reservations software. It is a good practice to “initialize” the software to a beginning state and open a window as well as using the following:

- Execute the split screen as the first command in the script to view smaller responses in a split screen environment. Execute the full screen command to view long responses in a full screen environment. You have to experiment with the length and type of responses Sabre will be returning to your screen. The more information you can display for the agent, the easier it is for them to execute the function you are scripting.
- Execute a “Clear All” command to clear any text present on the screen before the script executes. This ensures that the work area on the screen is not cluttered with unrelated text and is especially important when a script reads from the emulator.
- Precede each *Sabre* system command in the script with the {HOME} command. This ensures that the command entered by the script is executed on the top line of the active partition for both the full and split screen modes. Remember the cursor returns to the top of the accumulated responses when it is turned on.

Planning for Multiple Responses

Only use the WAIT statement for multiple Sabre system responses. If you use a WAIT statement when there is no second response, the script will wait two minutes before moving on.

Selecting the Correct Variable

- Minimize the different types of variables you use when creating scripts.
- If you make any modifications to the user-defined variables, remember that those changes apply to all scripts compiled using those variables.
- If you need to specify unique attributes to a commonly named variable in each script, use the DEFINE variable statement.
- Place variable statements at the top section of your script. This allows the DEFINE variable statements to initialize correctly before they are used in any other section of the program. This also allows you to quickly view the variable attributes of the DEFINE variable statements by looking at the top section of your script.

Creating Modular Scripts

Instead of creating one large script to accomplish a task, such as selling a hotel, create a variety of small scripts that do one specific task. If the small scripts are written to perform just a single task, they can be executed with the CHAIN or SPAWN command from many scripts.

An example of one large script is a script that would display a PNR, display hotel information, sell a hotel, and queue the finished PNR to an appropriate queue. As you create new custom scripts, you can collect several scripts of this type, with a great amount of duplication of coding from previous scripts. This adds to the size of each script and the total time needed to develop them.

A more economical type of scripting would be to separate the large script into several smaller scripts. The previous example would be divided into 3 scripts:

One to display the PNR

One to display hotel information and then sell the hotel

One to end and queue the PNR

With this approach, the three scripts would be only slightly larger then the one large script. The advantage is that you can re-use completed, working scripts when you begin new projects. For example, when you begin a new script to sell a car, two major portions of the script are already finished. The car script can SPAWN the Display PNR script, display car information and sell the car, then SPAWN the End and Queue PNR script. The only coding would be the commands to display the car information and sell the car, saving time and using memory efficiently. This type of scripting means that the coding to display a PNR is written once, instead of each time a newly created script needs to display a PNR.

Date and Time Calculations

Date and time values can be calculated using two basic operations: addition and subtraction. If any other operations are encountered by the compiler, an error message is displayed. Date values that are represented as “DDMMMYY” or “DDMMM” are supported for calculation. Calculations using dates are based on 1 day as the base value. Time values that are represented using the 12- or 24-hour clock are also supported for calculation (for example, 1P, 100P, 1300). Calculations using times are based on 1 second as the base value.

Date and time calculations allow you to design scripts that can perform some of the work that the agents have to execute. You can determine ticketing dates for early purchasing requirements, the number of nights a hotel room is needed, or the number of days a rental car is needed based on the arrival and departure dates within a specific city. This can all be done without prompting the agent to input more information.

To calculate dates, you can add a number of days to the variable representing the arrival date to find the departure date for a tour. You can subtract a whole number from the arrival date variable or you can subtract one date from another to find the elapsed days as in these examples:

```
[ELAPSED_DAYS]=[DATE1]-[DATE2]   ; elapsed days for this hotel
          ; reservation

[TOTAL_DAYS]=[DATE1]+[NUM_OF_DAYS]    ;elapsed days for this hotel
                    ;reservation
```

Time calculations are completed using similar formulas. However, you must remember that time calculations are based on 1 second, so you need to multiply your numbers to obtain results in minutes or hours. For example, this entry shows how to calculate elapsed, or total, time.

```
[ELAPSED_TIME]=[TIME1]-[TIME2]          ;elapsed time stated in
seconds

[TOTAL_TIME]=[TIME1]+[NUM_OF_SECONDS]
```

If you are calculating elapsed times, the result is displayed in total number of seconds. You must divide the result by 60 to convert that number to minutes. You divide total minutes by 60 to obtain the number of elapsed hours as in this example:

```
[ELAPSED_TIME]=[TIME1]-[TIME2]          ; elapsed time stated in
seconds

[ELAPSED_MINUTES]=([ELAPSED_TIME]/60)

[ELAPSED_HOURS]=([ELAPSED_MINUTES]/60)
```

The following is a more extensive example of date and time calculations.

```
DEFINE [DATE1 = DATE]
DEFINE [DATE2 + DATE]
DEFINE [TIME1 = TIME]
DEFINE [TIME2 = TIME]
DEFINE [ELAPSED_DAYS = N:2]
DEFINE [ELAPSED_TIME = N:6]
DEFINE [NEW_DATE = DATE]
DEFINE [NEW_TIME + TIME]

WINDOW
   Enter two dates.
          [DATE1]      [DATE2]
   Enter two times.
          TIME1]       [TIME2]
ENDWINDOW

DEFAULT [ELAPSED_DAYS] = [DATE2] - [DATE1]
DEFAULT [ELAPSE_TIME] = [TIME2] - [TIME1]
DEFAULT [NEW_DATE] = [DATE1] + 30
DEFAULT [NEW_TIME] = [TIME1] * 60 * 60
DEFAULT [DATE1] = [DATE1]
DEFAULT [TIME1] = [TIME1]
WINDOW
   There are [ELAPSED_DAYS] between the dates
   There is [ELAPSE_TIME] seconds between the time
   30 days from [DATE1] is [NEW_DATE]
   1 hour from [TIME1] is [NEW_TIME]
ENDWINDOW
```

In addition to calculating dates, you can also display the date information in a 12- or 24-hour format using the @SWITCHES system variable. Refer to “Using the @SWITCHES System Variable” on page 23 for more information.

24-hour time — Time of day as indicated on a 24-hour clock (0000 - 2359).

A

ASCII — Acronym for *American Standard Code for Information Interchange*. The acronym is pronounced “AS-kee.” A standard computer character set devised in 1968 to enable efficient data communication and to achieve compatibility among different computer devices.

C

cascading menu — A menu that appears when a cascading choice is selected. It contains a set of choices that are related to the cascading choice.

chevron — Symbols that indicate the start and stop of a *Sabre* command. Example: »

clipboard — A temporary storage area for text or graphics cut or copied from an application. You can paste the contents of the clipboard into an area in the same document or into another application. The clipboard keeps the information until you cut or copy another piece of text or a graphic to store on the clipboard. A clipboard is typically provided by the operating environment.

clone — The process of duplicating an existing variable.

command design — The mode you use to design the majority of your script.

compiler — An executable program that converts a script source file into an executable script.

conversion — The process of changing information from one form or representation to another. Do not use when referring to transferring data.

Cross of Lorraine — A *Sabre* system character used in many formats representing several possible functions.

cut/copy/paste — Allows you to copy or move text within *Sabre for Windows* desktop reservations software and the script, or to and from another application. Example: You can copy a fare calculation line from a WP entry or from an existing remarks line that the airline created. For details about using the clipboard, see Microsoft Windows helps.

D

dedicated fileserver — A fileserver that performs network functions only.

default — A value you specify to appear each time the value is needed. Some system shipped defaults are available when you install. You can change these defaults whenever you choose.

E

expression — A string of characters that the compiler evaluates.

K

keyword — A special word that identifies a particular type of statement or command, such as *IF* or *WINDOW*. Follow the capitalization style of the programming language involved.

L

logical operator — A symbol that allows you to combine two or more expressions and test them for validity.

loop — A group of instructions set off by a *MySabre Scribe* software command that executes in a repetitive manner.

M

menu — A list of action, routing, and settings choices. The types of menus are the menu bar, pull-down menu, cascaded menu, and pop-up menu.

P

paste — To place the contents of the clipboard—whatever was last cut or copied—at the insertion point or the standard location in a document by choosing the Paste command from a menu.

R

relational operator — A symbol that allows you to test an expression and test it for validity.

routine — A group of instructions that you want to isolate from the body of your script.

S

Sabre — The *Sabre* system is the computer that allows you to request availability, make bookings, and obtain travel information for your clients.

Sabre system format — A group of characters that, when typed on a *Sabre* system command line, allow the system to perform a specific function. For more information about *Sabre* system formats, refer to the *Sabre* Computer Reservations System Desktop Reference or F*FOX.

MySabre Scribe software — Allows the user to develop scripts.

screen design — The mode you use to design the windows for your script.

scripts — Scripts written for *Sabre* system formats.

scroll bar — A window component, associated with a scrollable area, that indicates to a user that more information is available or can be added in a particular direction and can be scrolled into view. The components of a scroll bar are the scroll box, shaft, and scroll buttons.

shortcut key — A key or combination of keys that a user can press to perform an action that is available from a menu.

special characters — Nonstandard characters that you can include in your script using a specific key sequence.

V

variable — A symbol that defines a value.