Siebel eScript Language Reference

Siebel Innovation Pack 2013
Version 8.1/8.2
September 2013

ORACLE

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

1 What's New in This Release

No new features have been added to this guide for this release. This guide has been updated to reflect only product name changes.

What's New in Siebel eScript Language Reference, Version 8.1, Rev A and Version 8.2

Table 1 lists changes in this version of the documentation to support this release of the software.

Table 1. What's New in Siebel eScript Language Reference, Version 8.1, Rev A and Version 8.2

Topic	Description
"About Local and Global Variables" on page 44	New topic. Siebel eScript includes local and global variables. You declare these variables differently. Access to these variables also varies.
"Using a Local Variable Is Preferable to Using a Global Variable" on page 45	New topic. It is recommended that you use a local variable where possible instead of a global variable.
"Declaring a Variable In a Statement Block" on page 47	New topic. If you declare a variable in a statement block in a method, then you can reference that variable anywhere in the method.
"Running Browser Script When Siebel CRM Starts a Siebel Application" on page 50	New topic. You can configure Siebel CRM to run Browser Script when it starts a Siebel application.
"For In Statement" on page 63	Modified topic. The DONT_ENUM attribute is a predefined attribute that you cannot modify.
"Using the Throw Statement with Nested Try Catch Blocks" on page 69	New topic. To handle an exception, you can use the Throw statement with nested Try catch blocks.
"Array Methods" on page 76	Modified topic. The following methods were added: Concatenate Array, Get Subarray, Shift Array Left, and Shift Array Right.
"Example" on page 113	New topic. To avoid receiving an unhandled exception error, you can use the RaiseErrorText method or the RaiseError method instead of the Throw statement.
"Setting the Day to a Value That Exceeds 31" on page 133	New topic. You can write code that adds any number of days to a date. Siebel eScript automatically converts the number of days to the correct month and year.

Table 1. What's New in Siebel eScript Language Reference, Version 8.1, Rev A and Version 8.2

Topic	Description
"Using the Dispatch Identifier to Call a COM Method" on page 151	New topic. To use the DISPID (Dispatch Identifier) of a COM method to call that COM method, you make an IDispatch::Invoke call.
"Using a Multivalue List to Avoid Unexpected Rounding" on page 158	New topic. If you must use a value that exceeds 2^{53}, then it is recommended that you use a calculated field that uses the sum of a multivalue list instead of using Siebel eScript.
"Preprocessing Error Messages" on page 320	Modified topic. You can organize constants and other definitions in include files, and then use #include directives to add these definitions to any source file.

Additional Changes

This version of this book includes revised code that fixes a problem in the old code that caused an Undefined Object error. For more information, see "The Arguments Property of a Function" on page 31.

This version of Siebel eScript Language Reference also includes structural changes to the content, such as topic organization and heading arrangement.

2 About Siebel eScript

This chapter describes Oracle's Siebel eScript. It includes the following topics:

Overview of Siebel eScript

Siebel eScript is a programming language that is syntactically and semantically compatible with JavaScript. It includes an editor, debugger, interpreter, and compiler. It runs on the Windows and UNIX operating systems.

JavaScript is typically part of a Web browser and can run while the user is connected to the Internet. Siebel eScript is part of Siebel CRM. The Siebel Application Object Manager interprets it at run time. You do not require a Web browser to run it. It provides access to the hard disk and other parts of the Siebel client or Siebel Server. ECMAScript does not provide this access.

Siebel Tools allows you to configure Siebel CRM without scripting. It is recommended that you use Siebel eScript only after you determine that you cannot use any other tool. For more information, see the chapter about using Siebel eScript in *Siebel Object Interfaces Reference* on *Siebel Bookshelf*.

About Siebel eScript Code

Siebel CRM uses two versions of Siebel eScript code:

- **T eScript code.** Compiled by the original T eScript engine.
- **ST eScript code.** Compiled by the ST eScript engine beginning with Siebel CRM version 8.0. Siebel CRM introduced this code in Siebel CRM version 7.8. ST eScript code is the default code beginning with Siebel CRM version 8.0.

This book mentions these code versions only if differences exist between how the code functions.

Features of ST eScript Code

ST eScript code includes the following features. T eScript code does not include these features:

- **Strong typing of variables.** Sets the data type of a variable when you declare the variable. For more information, see "Using Strongly Typed and Typeless Variables" on page 43.
- **Script Assist utility.** For more information, see "About the Script Assist Utility" on page 18.
- **Fix and Go feature.** Allows you to edit and debug a script without recompiling it.

- **ToolTip feature.** Displays method signature data after you enter a method name in the Script Editor.
- **Script profiling.** For more information, see "Monitoring the Performance of Your Script" on page 51.

For more information about these features, see *Using Siebel Tools*.

Compatibility Between ST eScript Code and T eScript Code

ST eScript code is mostly backward compatible with T eScript code. Table 2 describes features that are not backward compatible.

Table 2. Incompatibilities Between ST eScript Code and T eScript Code

Feature	Description
Strong typing	For more information, see "Using Strongly Typed and Typeless Variables" on page 43.
Comparison operations	ST eScript code does the following: ■ If it compares typeless variables, then it compares object values. ■ If it compares strongly typed variables, then it compares object identities. If you are not aware of these differences, then the results of a comparison operation that involves strongly typed variables might be misleading. For more information, see "Using Logical Operators and Conditional Expressions" on page 35.
Implicit variable type conversion	Siebel eScript does an implicit conversion differently depending on if a variable is strongly typed or if it is typeless. For more information, see "How Siebel eScript Converts Data Types" on page 24.
Methods	Siebel eScript restricts the parameters in some methods differently depending on if you use ST eScript code or T eScript code. If it restricts these parameters differently, then this book describes those differences. For an example, see "Set Array Length Method" on page 153.
Properties	For more information, see "Overview of Regular Expression Methods" on page 194.

Commands	ST eScript code does not support the #Define statement or the #If statement. As an alternative, you can use a Var statement. For example, consider the following code:
	`#define MY_DEFINE "abc"`
	The following code replaces the #Define statement with the Var statement:
	`var MY_DEFINE = "abc";`

Objects and arrays	For more information, see "Referencing Objects and Arrays" on page 17.

Referencing Objects and Arrays

If you write code that references an array item, an object function, or object data, and:

- **You use T eScript code.** Siebel eScript automatically creates a new object.
- **You use ST eScript code.** Siebel eScript does not automatically create a new object. If you use ST eScript code, then you must configure Siebel eScript to explicitly initialize the object.

Referencing Object Functions or Data

The following script runs correctly using T eScript code but fails at runtime using ST eScript code:

```
var oArr = new Array ();
oArr[0].m_Data =1;
```

You must configure Siebel eScript to initialize the data object that it references so that the script runs correctly with ST eScript code. For example:

```
var oArr = new Array ();
oArr[0] = new Object ();
oArr[0].m_Data =1;
```

Using Arrays

The following script runs correctly using T eScript code but fails at runtime using ST eScript code:

```
var oArr = new Array ();
oArr[2][3].m_Data = 2;
```

You must configure Siebel eScript to initialize the data object that it references so that the script runs correctly with ST eScript code. For example:

```
var oArr = new Array ();
oArr[2] = new Array ();
oArr[2][3] = new Object ();
oArr[2][3].m_Data = 2;
```

Reverting ST eScript Code to T eScript Code

If you use ST eScript code, then you can revert this code back to T eScript code. This technique is not recommended.

To revert ST eScript code to T eScript code

1. Create a service request or call Oracle Global Customer Support.

   For help with reverting ST eScript code to T eScript code, create a service request (SR) on My Oracle Support. Alternatively, you can phone Oracle Global Customer Support directly to create a service request or get a status update on your current SR. Support phone numbers are listed on My Oracle Support.

2. In Siebel Tools, turn off the ST eScript engine.

   For more information, see *Using Siebel Tools*.

3 Remove any script that you typed prior to compiling with the ST eScript engine.

4 Do a full compile of all objects in the repository.

About the Script Assist Utility

The *Script Assist utility* is a feature in Siebel Tools that helps you write ST eScript code. Starting with Siebel CRM version 8.0, Script Assist can access information about object definitions in the Siebel Repository File (SRF) that your script references, and then display this information in the Script Assist window.

Script Assist limits the choices you can make according to the information that the SRF contains. This feature helps prevent scripting errors and simplifies the scripting process. For example, it prevents you from writing code that causes Siebel CRM to write to a read-only field, or to get a value from a field that does not exist in the object. It displays the required and optional parameters for the following items:

- Siebel methods
- Global methods
- Global functions
- Custom functions
- Methods available for InvokeMethod calls

To identify the object that your script references, Script Assist can use the following object reference key word in a script and display the appropriate fields for this object:

```
this
```

For more information about Script Assist, see *Using Siebel Tools*.

Table 3 lists the objects and methods that Script Assist can access in the SRF.

Table 3. Objects and Methods That Script Assist Can Access in the SRF

Object	Method
Applet	Script Assist can access the following methods: ■ BusComp ■ BusObject
Application	Script Assist can access the following methods: ■ ActiveBusObject ■ ActiveViewName ■ GetBusObject ■ GetService

Table 3. Objects and Methods That Script Assist Can Access in the SRF

Object	Method
Business Component	Script Assist can access the following methods: ■ ActivateField ■ ActivateMultipleFields ■ Associate ■ BusObject ■ DeActivateFields ■ GetAssocBusComp ■ GetFieldValue ■ GetFormattedFieldValue ■ GetMultipleFieldValues ■ GetMvgBusComp ■ GetPickListBusComp ■ GetViewMode ■ ParentBusComp ■ SetFieldValue ■ SetFormattedFieldValue ■ SetMultipleFieldValues
Business Object	GetBusComp
Business Service	InvokeMethod

About Data Types and Numbers

This topic describes data types in Siebel eScript. It includes the following topics:

For more information, see "Determining the Data Type of a Variable" on page 47.

About Primitive Data Types

A *primitive data type* is a type of data that Siebel eScript provides as a fundamental part of the code. It does not possess other properties or functions.

The bool, chars, and float data types must use lowercase.

Bool Data Type

The bool data type defines and manipulates a Boolean object. A Boolean value is true or false.

Chars Data Type

The chars data type defines and manipulates a string. A *chars value* is a sequence of alphanumeric characters. It can include any sequence of 16-bit, unsigned integers.

Float Data Type

The float data type defines and manipulates a floating point number.

Integer is not a Siebel eScript data type. You can write code that uses a float variable. Some code that expects an integer variable converts a float variable to an integer.

Undefined Data Type

If Siebel CRM saves nothing in a variable, then it is undefined. An undefined variable occupies space until Siebel CRM saves a value in it. When Siebel CRM saves a value in a variable, it modifies the type of the variable according to the value.

The following example determines if a variable is undefined:

```
var test;
if (typeof test == "undefined")
TheApplication().RaiseErrorText("test is undefined");
```

About Composite Data Types

This topic describes the composite data types that Siebel eScript uses. A *composite data type* is a complex type of data that can include properties and functions.

Overview of the Object Data Type

The ECMAScript standard uses the following description for an object:

> A member of the type Object. It is an unordered collection of properties, each of which includes a primitive value, object, or function. A function stored in a property of an object is referred to as a method.

Siebel eScript does not use a class hierarchy that conforms to this ECMAScript standard. Instead, it instantiates an object in the following ways:

- As an Object type
- From an object that it instantiates as an Object type

These objects are new object types that Siebel eScript can use to instantiate other objects. Each object includes an implicit constructor function that it implements through the following command:

```
new
```

You can configure Siebel eScript to add properties dynamically to an object. An object inherits all the properties of the objects that reside in the ancestral chain of the object.

The object type known as Object is a generic object type. If you declare an object as an Object type, then it does not inherit properties from any object.

For more information, see "Use a Primitive Data Type Instead of an Object Data Type" on page 55.

Boolean Data Type

The value of a Boolean object is a bool value, which is true or false. It is a property of the Boolean object. If you use a Boolean variable in a numeric context, then Siebel eScript does the following conversion:

- If the value of a bool variable is true, then it converts this value to 1.
- If the value of a bool variable is false, then it converts this value to 0.

To create a Boolean object, you use the Boolean constructor in the type of expression:

```
new
```

String Data Type

The string value is a chars value. Siebel eScript adds it as a property of the String object. A pair of double or single quotation marks brackets a string. For example:

```
"I am a string"
'so am I'
"344"
```

In this example, the 344 string is an array of characters. The number 344 is a value that Siebel eScript can use in a numeric calculation.

To create a string data type, you use the String constructor in the following type of expression:

```
new
```

Siebel eScript does one of the following, depending on the context:

- Converts a string to a number
- Converts a number to a string

For more information, see "How Siebel eScript Converts Data Types" on page 24.

Number Data Type

The value of a number is a float value. It is a property of the Number object.

To create a number object, you use the Number constructor in the following type of expression:

```
new
```

For more information, see "About Numbers" on page 26.

Array Data Type

An array is a series of data that Siebel eScript stores in a variable. Each datum is associated with an index number or string. The following example illustrates how Siebel eScript stores data in an array:

```
var Test = new Array;
Test[0] = "one";
Test[1] = "two";
Test[2] = "three";
```

In this example, the Test variable is an array that includes three strings. You can write code that uses an array variable as one unit. To reference a string individually, you can append the bracketed index of the element after the array name.

To reference a property:

- An array uses an index.
- An object uses a property name or a method name.

For more information, see "Array Methods" on page 76.

Null Data Type

The null object indicates that a variable is empty. It does not contain a value, although it might have previously contained a value. The following term identifies a null data type:

```
null
```

The following keyword allows you to compare a value to a null object:

```
null
```

Null includes a literal representation. The following example is valid:

```
var test = null;
```

Siebel eScript can compare any variable that contains a null value to a null literal.

Other Object Types That Siebel eScript Supports

Table 4 lists other object types that Siebel eScript supports.

Table 4. Other Object Types That Siebel eScript Supports

Object	Description
Application	For more information, see *Siebel Object Interfaces Reference*.
BLOB	For more information, see "BLOB Methods" on page 101.
BLOB Descriptor	For more information, see "About the BLOB Descriptor" on page 102.
Buffer	For more information, see "Buffer Methods" on page 108.
Business Component	For more information, see "Overview of Date Methods" on page 122.
Business Object	For more information, see *Siebel Object Interfaces Reference*.
CfgItem	The CfgItem object is a Siebel Product Configuration object.
Clib	For more information, see "Overview of the Clib Object" on page 215.
CTIData	For more information, see *Siebel Communications Server Administration Guide*.
CTIService	For more information, see *Siebel Communications Server Administration Guide*.
Date	For more information, see "Overview of Date Methods" on page 122.
Exception	For more information, see "Function Object" on page 177.
File	For more information, see "Clib Open File Method" on page 222.
Math	For more information, see "Mathematical Methods" on page 179.
Property Set	For more information, see *Siebel Object Interfaces Reference*.
Regular Expression	For more information, see "Regular Expression Methods" on page 193.
SELib	For more information, see "Siebel Library Methods" on page 201.
Service	For more information, see *Siebel Object Interfaces Reference*.
Web Applet	For more information, see *Siebel Object Interfaces Reference*.

Properties and Methods of Common Data Types

Common data types include properties and methods that you can use with any variable of this type. Any string variable can use any string method. Examples of common data types include a number or string. For example, assume you use a numeric variable named VariableA and you must convert it to a string. The following example illustrates how you can use the toString method to convert a numeric variable to a string:

```
var VariableA = 5;
var VariableB = num.toString();
```

After this code finishes, VariableA contains the number 5 and VariableB contains the string 5.

The following methods are common to variables:

- **ValueOf method.** Returns the value of a variable. Value is an implicit property of some objects, including number, string, and Boolean objects.
- **ToString method.** Returns the value of a variable that is expressed as a string. Value is an implicit property of number and Boolean objects.

How Siebel eScript Converts Data Types

Siebel eScript implicitly converts data types in many mixed-type contexts. You must use conversion methods to make sure your code does the required conversions. For more information, see "Conversion Methods" on page 154.

Concatenation Can Cause a Conversion

Siebel eScript converts the data type of a typeless variable in the following situations:

- If you write Siebel eScript code that subtracts a string from a number, or that subtracts a number from a string, then it converts this string to a number and subtracts the two values.
- If you write Siebel eScript code that adds a string to a number, or that adds a number to a string, then it converts this number to a string and concatenates the two strings.

Siebel eScript must always convert a string to a base 10 number. This string must contain only digits. For example, the following string does not convert to a number because Text is meaningless as part of a number in Siebel eScript:

```
110Text
```

The following examples result in Siebel eScript doing a conversion:

```
s = "dog" + "house"   // s = "doghouse", two strings are concatenated.
t = "dog" + 4         // t= "dog4", a number is converted to a string
u = 4 + "4"           // u = "44", a number is converted to a string
v = 4 + 4             // v = 8, two numbers are added
w = 23 - "17"          // w = 6, a string is converted to a number
```

Using a Conversion Method

You must use a conversion method to make sure Siebel eScript does conversions when it adds, subtracts, or does other arithmetic operations. The following example uses a conversion method to convert a string input to a numeric value:

```
var n = "55";
var d = "11";
var division = Clib.div(ToNumber(n), ToNumber(d));
```

Use can use the parseFloat method of the global object to specify a more stringent conversion. For more information, see “Convert String to Floating-Point Number Method” on page 156.

You must use a conversion method in situations where Siebel eScript does not do a conversion. Siebel eScript includes many global methods that convert data types. For more information, see “Conversion Methods” on page 154.

Setting the Data Type Can Cause a Conversion

Siebel eScript does conversions differently depending on if the variable is typeless or strongly typed. For more information, see “Using Strongly Typed and Typeless Variables” on page 43.

How Siebel eScript Converts a Typeless Variable

If Siebel eScript sets the data type for a typeless variable, then it converts this variable only to another typeless variable. For example, the following examples result in Siebel eScript converting VariableA to a string:

```
var VariableA = 7.2;
var VariableB = “seven point 2”
VariableA = VariableB;
```

How Siebel eScript Converts a Strongly Typed Variable

Table 5 describes how Siebel eScript converts a strongly typed variable. In this table, assume that Siebel eScript must convert VariableA to VariableB.

Table 5. How Siebel eScript Converts a Strongly Typed Variable

VariableA Type	VariableB Type								
	Value	**Chars**	**Bool**	**Float**	**Object**	**String**	**Number**	**Boolean**	**Other**
Value	Same	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Chars	Yes	Same	Yes	Yes	Yes	Yes	Yes	Yes	Yes W
Bool	Yes	Yes	Same	Yes	Yes	Yes	Yes	Yes	Yes
Float	Yes	Yes, W	Yes	Same	Yes, W	Yes, W	Yes	Yes	Yes, W
Object	Yes	Err	Err	Err	Same	None	None	None	None
String	Yes	Yes	Err	Err	Err	Same	Err	Err	Err
Number	Yes	Err	Err	Yes	Err	Err	Same	Err	Err
Boolean	Yes	Err	Yes	Err	Err	Err	Err	Same	Err
Other	Yes	Err	Err	Err	Err	Err	Err	Err	Same

Table 5 on page 25 uses the following abbreviations:

- **Yes.** Siebel eScript converts the variable.

- **W.** Siebel Tools might display a message when it compiles the script. This message warns that the conversion might not occur. The warning and conversion depend on the properties of the variables that are involved when Siebel eScript sets the data type.
- **Err.** A compilation error occurs.
- **None.** No conversion is required. A conversion is not typically required to modify an Object variable to a specialized object type.
- **Same.** VariableA and VariableB are of the same type.
- **Value.** Indicates a typeless variable. It describes the conversion that Siebel eScript does in the following situations:
  - Convert a strongly typed variable to a typeless variable.
  - Convert a typeless variable to a strongly typed variable.
- **Other.** Indicates predefined types and custom types that are not the following types:
  - Object
  - String
  - Number
  - Boolean

About Numbers

This topic describes the types of numbers that you can use with Siebel eScript. Siebel eScript treats a number that contains a character other than a decimal point as a string. For example, the number 100,000 is a string, including the comma. The exception is hexadecimal numbers and scientific notation.

The number types that this topic describes are not data types. You cannot write code that uses one of these number types as a data type in the declaration of a strongly typed variable. For more information, see "Using Strongly Typed and Typeless Variables" on page 43.

Integer Numbers

An *integer number* is a positive whole number, a negative whole number, or zero. Siebel eScript recognizes the following:

- An integer constant or an integer literal in decimal, hexadecimal, or octal notation.
- A decimal constant or a decimal literal in decimal representation.

You cannot write code that strongly types a variable as an integer. You can write code that uses the float primitive or the value of the float primitive as an integer. For more information, see "Using Strongly Typed and Typeless Variables" on page 43.

Hexadecimal Numbers

A *hexadecimal number* is a number that uses base 16 digits. It uses digits from the following sets:

- 0 through 9
- A through F
- a through f

The following format precedes a hexadecimal number:

0x

A hexadecimal number is not case-sensitive in Siebel eScript.

Table 6 lists example hexadecimal numbers and their decimal equivalents.

Table 6. Example Hexadecimal Numbers and Their Decimal Equivalents

Hexadecimal Number	Decimal Number
0x1	1
0x01	1
0x100	256
0x1F	31
0x1f	31
0xABCD	43981
var a = 0x1b2E	var a = 6958

Octal Numbers

An *octal number* is a number that uses base 8 digits. It includes digits from the following set:

0 through 7

A zero precedes an octal number.

Table 7 lists example octal numbers and their decimal equivalents.

Table 7. Example Octal Numbers and Their Decimal Equivalents

Octal Number	Decimal Number
00	0
05	5
077	63
var a = 0143	var a = 99

Floating Point Numbers

A *floating-point number* is a number that includes a whole part and a fractional part. A decimal separates these parts. For example, 10.33. Some developers refer to a floating-point number as a float. The float data type is not a floating-point number. For more information, see "Float Data Type" on page 20.

For more information, see "Preventing a Floating-Point Error" on page 49.

Floating Decimal Numbers

A *floating decimal number* is a number that uses the same digits as a decimal integer but uses a period to indicate the fractional part of the number. For example:

```
0.32, 1.44, 99.44
var a = 100.55 + .45;
```

Scientific Numbers

A *scientific number* is a number that uses decimal digits and exponential notation. The following items represent exponential notation:

- e
- E

A scientific number is useful if you must use very large or very small numbers. Scientific notation is also known as exponential notation.

Table 8 lists example scientific numbers and their decimal equivalents.

Table 8. Example Scientific Numbers and Their Decimal Equivalents

Scientific Number	Decimal Number
4.087e2	408.7
4.087E2	408.7
4.087e+2	408.7
4.087E-2	0.04087
var a = 5.321e31 + 9.333e-2	var a = 532100000000000000000000000000 + 0.09333

NaN Numbers

NaN is a value that is an abbreviation for the following phrase:

not a number

NaN is not a data type. NaN does include a literal representation. To test for NaN, you must use the isNaN function. The following example illustrates this usage:

```
var Test = "Test String";
if (isNaN(parseInt(Test)))
TheApplication().RaiseErrorText("Test is Not a Number");
```

If the parseInt function attempts to parse Test String into an integer, then it returns NaN because Test String does not represent a number.

Numeric Constants

You can write code that references a numeric constant as a property of the Number object. A numeric constant does not include a literal representation.

Table 9 describes some numeric constants.

Table 9. Numeric Constants in Siebel eScript

Numeric Constant	Value	Description
Number.MAX_VALUE	1.7976931348623157e+308	Largest positive number.
Number.MIN_VALUE	2.2250738585072014e-308	Smallest positive nonzero value.
Number.NaN	NaN	Not a number.
Number.POSITIVE_INFINITY	Infinity	Number greater than MAX_VALUE.
Number.NEGATIVE_INFINITY	-Infinity	Number less than MIN_VALUE.

About Functions and Methods

A *function* is an independent section of code that does the following:

1. Receives information
2. Performs some action on this information
3. Returns a value to the item that called it

It begins with the following statement:

```
Function functionname
```

It ends with the following statement:

```
End Function
```

You use the same format as you use with a variable to name a custom function. You can write code that uses any valid variable name as a function name. It is recommended that you use a name that describes the work that the function performs.

You can write code that calls a function repeatedly from various objects or script. It is similar to a subroutine. To call a function, you must know what information the function requires as input and what information it provides as output. This book describes the predefined functions that come with Siebel eScript. You can write code that uses these functions any time you use the Siebel eScript interpreter.

You can write code that uses a function anywhere that you can use a variable. To use a function, you do the following:

- To declare it, you use the function keyword.
- To determine the data that Siebel eScript must pass to the function, you include the function operator. To include this operator, you use a pair of parentheses immediately after the function name. For example:

```
TheApplication.RaiseErrorText()
```

A *Siebel VB method* is a function that is part of an object or class. It can include a predefined procedure that you can use to call a function that does an action or a service for an object.

A *Siebel VB statement* is a complete instruction.

For more information, see Passing a Value to a Function on page 48.

Example of a Function

The TheApplication.RaiseErrorText function is an example of a function that allows you to display formatted text if an error occurs. It does the following work:

- Receives a string from the function that calls it
- Displays this string in an alert box in the Siebel client
- Stops the script

About Function Scope

A function is global. Siebel eScript can call it from anywhere in a script in the object where you declare it. The examples in this topic achieve the following results:

- The code passes the 3 and 4 literals as parameters to the SumTwo function.
- The SumTwo function includes corresponding a and b variables. These variables contain the literal values that Siebel eScript passes to the function.

Example of Calling a Function as a Function

The following example calls the SumTwo function:

```
function SumTwo(a, b)
{
   return (a + b)
}

TheApplication().RaiseErrorText(SumTwo(3, 4));
```

Example of Using a Method to Call a Function

The following example uses a method of the global object to call the SumTwo function:

```
function SumTwo(a, b)
{
   return (a + b)
}

TheApplication().RaiseErrorText(global.SumTwo(3, 4));
```

The Arguments Property of a Function

The *arguments* property of a function is a list of the arguments that Siebel eScript passes to the function. The first argument is arguments[0], the second argument is arguments[1], and so on. You can write code that references the arguments property for a function only in that same function.

You can configure a function that includes an indefinite number of arguments. The following example uses a variable number of arguments, and then returns the sum:

```
function SumAll()
{
   var total = 0;
   for (var ssk = 0; ssk < arguments.length; ssk++)


{
      total += arguments[ssk];
}
   return total;
}
```

About Recursive Functions

A *recursive function* is a type of function that calls itself or that calls another function that calls the first function. Siebel eScript allows recursion. Each call to a function is independent of any other call to this function. If a function calls itself too many times, then the script runs out of memory and aborts.

In the following example, the factor function calls itself so that it can factor a number. Factoring is an appropriate use of recursion because it is a repetitive process where the result of one factor uses the same rules to factor the next result:

```
function factor(i) //recursive function to print factors of i,
{// and return the number of factors in i
   if ( 2 <= i )
{
      for ( var test = 2; test <= i; test++ )
{
         if ( 0 == (i % test) )
{
// found a factor, so print this factor then call
// factor() recursively to find the next factor
         return( 1 + factor(i/test) );
}
}
}
```

```
// if this point was reached, then factor not found
   return( 0 );
}
```

Error Checking with Functions

If a function fails, then some functions return a special value. Consider the following example:

- To allow a script to read from or write to a file, the Clib.fopen method opens or creates this file.
- If the script calls the Clib.fopen method but this method cannot open a file, then the method returns null.
- If the script then attempts to read from or write to this file, then Siebel CRM creates an error.

To prevent this error, you can use the following code to determine if Clib.fopen returns null when it attempts to open a file. If it does return null, then you can write code that aborts the script and displays an error message:

```
var fp = Clib.fopen("myfile.txt", "r");
var fp = Clib.fopen("myfile.txt", "r");
if (null == fp)//make sure null is not returned
{
   TheApplication().RaiseErrorText("Error with fopen as returned null " +
   "in the following object: " + this.Name() + " " + e.toString() + e.errText());
}
```

For more information, see “Overview of the Clib Object” on page 215.

Where Data Resides

Data in a script resides in a literal or in a variable. The following example includes variables and literals:

```
var TestVar = 14;
var aString = "test string";
```

This code does the following:

- Saves the following literal value to the TestVar variable:

  14

- Saves the following literal value to the aString variable:

  test string

After you save a literal value in a variable, you can reference this variable anywhere in the script where you declare the variable.

3 Using Siebel eScript

This chapter describes how to use Siebel eScript. It includes the following topics:

Using Operators in Siebel eScript

This topic describes operators you can use in Siebel eScript. It includes the following topics:

Overview of Mathematical Operators

Table 10 describes the basic arithmetic operators you can use in Siebel eScript.

Table 10. Basic Arithmetic Operators in Siebel eScript

Operator	Description
=	Make one number equal to another number.
+	Add two numbers.
-	Subtract one number from another number.
*	Multiply two numbers.
/	Divide one number by another number.
%	Return a remainder after a division operation. It is a modulo.

The following examples use variables and arithmetic operators:

```
var i;
i = 2;     //i is now 2
i = i + 3; //i is now 5, (2 + 3)
i = i - 3; //i is now 2, (5 - 3)
i = i * 5; //i is now 10, (2 * 5)
i = i / 3; //i is now 3.333..., (10 / 3)
i = 10;    //i is now 10
i = i % 3;  //i is now 1, (10 mod 3)
```

If uncertainty exists about how Siebel eScript might evaluate an expression, then it is recommended that you use parentheses. This recommendation is true even if you do not require parentheses to modify the sequence that Siebel eScript uses to evaluate expressions.

Using a Shortcut Operation to Do an Arithmetic Operation

A *shortcut operation* is a combination of the equal operator (=) with another operator. You can use a shortcut operation to reduce the amount of code you write. This technique does the following:

- To do an operation on the value that occurs to the left of the equal operator, it uses the value that occurs to the right of the equal operator.
- Saves the result in the value that occurs to the left of the equal operator.

To use a shortcut operation to do an arithmetic operation

- Add an operator immediately before the equal operator.

  For example, instead of writing i = i + 3, you can write i += 3.

Table 11 describes the shortcut operators that you can use in Siebel eScript.

Table 11. Shortcut Operators You Can Use in Siebel eScript

Shortcut Operation	Description
+=	Add a value to a variable.
-=	Subtract a value from a variable.
*=	Multiply a variable by a value.
/=	Divide a variable by a value.
%=	Return a remainder after a division operation.

The following example uses shortcut operators:

```
var i;
i += 3; //i is now 5 (2 + 3). Same as i = i + 3.
i -= 3; //i is now 2 (5 - 3). Same as i = i - 3.
i *= 5; //i is now 10 (2 * 5). Same as i = i * 5.
i /= 3; //i is now 3.333...(10 / 3). Same as i = i / 3.
i = 10; //i is now 10
i %= 3;  //i is now 1, (10 mod 3). Same as i = i % 3.
```

Modifying the Sequence That Siebel eScript Uses to Evaluate an Expression

Siebel eScript evaluates the operators in an expression in the following order:

1. Arithmetic operators
2. Comparison operators
3. Logical operators

You can write code that modifies this order.

To modify the sequence that Siebel eScript uses to evaluate an expression

- Use parentheses to group operations.

Siebel eScript performs operations in parentheses before it performs operations that occur outside of parentheses. It performs multiplication operations and division operations in an expression before it performs addition operations and subtraction operations. You can use parentheses to modify this sequence.

Table 12 includes an example of how Siebel eScript calculates a grouped expression.

Table 12. Example of Calculating a Grouped Expression

No Grouping	Equivalent	Not Equivalent
4 * 7 - 5 * 3 = 13 Siebel eScript calculates this expression as 28 - 15 = 13.	(4 * 7) - (5 * 3) = 13 Siebel eScript calculates this expression as 28 - 15 = 13.	4 * (7 - 5) * 3 = 24 Siebel eScript calculates this expression as 4 * 2 * 3 = 24.

Using Logical Operators and Conditional Expressions

Note the following:

- A *logical operator* is a type of operator that compares two values, and then determines if the result is true or false. A variable or any other expression can include true or false.
- A *conditional expression* is an expression that does a comparison.

You can use a logical operator to make a decision about the statements that reside in a script to run according to how Siebel eScript evaluates a conditional expression.

Table 13 describes the logical operators that you can use in Siebel eScript.

Table 13. Logical Operators That You Can Use in Siebel eScript

Logical Operator	Description
!	**Not.** Reverse of an expression. If (a+b) is true, then !(a+b) is false.
&&	**And.** If the value of every expression in the statement is true, then the entire statement is true. For example, if the first expression is false, then Siebel eScript does not evaluate the second expression because the entire expression is false.
||	**Or.** If the value of one expression in the statement is true, then the entire statement is true. For example, if the first expression is true, then Siebel eScript does not evaluate the second expression because the entire expression is true.
==	**Equality.** If the values of all expressions in the statement are equal to each other, then the entire statement is true. If the value of one expression is not equal to the value of any other expression, then the entire statement is false. **CAUTION:** The equality operator (==) is different from the assignment operator (=). If you use the assignment operator to test for equality, then your script fails because Siebel eScript assigns the right hand side of the expression to a variable that resides on the left hand side of this expression. For more information, see "Using the Equality Operator with a Strongly Typed Variable" on page 37.
!=	**Inequality.** If the value of one expression is different from the value of any other expression, then the entire statement is true. If the value of all expressions in the statement are equal, then the entire statement is false.
<	**Less than.** If the expression is a < b, and if a is less than b, then the statement is true.
>	**Greater than.** If the expression is a > b, and if a is greater than b, then the statement is true.
<=	**Less than or equal to.** If the expression is a <= b, and if a is less than or equal to b, then the statement is true.
>=	**Greater than or equal to.** If the expression is a >= b, and if a is greater than or equal to b, then the statement is true.

Example of Using Logical Operators and Conditional Expressions

Assume you design a simple guessing game where you configure Siebel CRM to choose a number between 1 and 100, and the user attempts to guess the value of this number. The game provides feedback if the user is correct or if the user answer is higher or lower than the number that the game chooses. The following Siebel eScript code implements this guessing game. Assume that the GetTheGuess function is a custom function that gets the guess:

```
var guess = GetTheGuess(); //get the user input, which is 1, 2, or 3
target_number = 2;
if (guess > target_number)
{
   TheApplication().RaiseErrorText("Guess is too high.");
}
if (guess < target_number)
{
   TheApplication().RaiseErrorText("Guess is too low.");
}
if (guess == target_number);
{
   TheApplication().RaiseErrorText("You guessed the number!");
}
```

In this example, the action that Siebel eScript performs depends on if the value in the parenthesis in an If statement is true or false:

- **True.** It runs the statement block that follows this If statement.
- **False.** It ignores the statement block that follows this If statement and runs the script that occurs immediately after the statement block.

Using the Equality Operator with a Strongly Typed Variable

If ST eScript code does an equality operation, then it compares different objects depending on the following types of variables that are involved in the comparison:

- **Typeless variable.** It compares object values.
- **Strongly typed variable.** It compares object identities.

For more information, see "Using Strongly Typed and Typeless Variables" on page 43.

Example of Using the Equality Operator with Strongly Typed Variables

The comparison in the following example involves strongly typed variables. The result is always not equal because Siebel eScript compares object identities in this example. It does not compare object values:

```
function foo ()
{
  var oStr1 : String = new String ("aa");
  var oStr2 : String = new String ("aa");
  if (oStr1 == oStr2)
```

```
      TheApplication ().RaiseErrorText ("equal");
  else
      TheApplication ().RaiseErrorText ("not equal");

}
```

Example of Using the Equality Operator with Typeless Variables

The result of the comparison in the following example is always not equal. The variables are typeless. The String is an object and Siebel eScript does object comparisons in the If statement:

```
function foo ()
{
  var oStr1 = new String ("aa");
  var oStr2 = new String ("aa");
  if (oStr1 == oStr2)
     TheApplication ().RaiseErrorText ("equal");
  else
     TheApplication ().RaiseErrorText ("no equal");
}
```

Making Sure Siebel eScript Compares Variable Values in an Equality Operation

This topic describes how to make sure Siebel eScript compares the values of variables that reside in an equality operation.

To make sure Siebel eScript compares variable values in an equality operation

- Use the valueOf method.

  For example:

  ```
  function foo ()
  {
   var oStr1 = new String ("aa");
   var oStr2 = new String ("aa");
   if (oStr1.valueOf () == oStr2.valueOf ())
      TheApplication ().RaiseErrorText ("equal");
   else
      TheApplication ().RaiseErrorText ("no equal");
  }
  ```

- Use primitive data types.

  For example:

  ```
  function foo ()
  {
   var oStr1 : chars = "aa"
   var oStr2 : chars = "aa";
   if (oStr1 == oStr2)
      TheApplication ().RaiseErrorText ("equal");
   else
      TheApplication ().RaiseErrorText ("no equal");
  }
  ```

Increasing or Decreasing the Value of a Variable

You can use the increment or decrement operator in the following ways:

- **Before a variable.** Siebel eScript modifies the variable before it uses it in a statement.
- **After a variable.** Siebel eScript modifies the variable after it uses it in a statement.

These operators add or subtract 1 from a value. For example, i++ is shorthand for i = i + 1.

To increment or decrement a variable

- To add 1 to a variable, you use the following operator:

  ```
  ++
  ```

- To subtract 1 from a variable, you use the following operator:

  ```
  --
  ```

The following example uses increment and decrement operators:

```
var i;
var j;
i = 4; //i is 4
j = ++i; //j is 5 and i is 5. Siebel eScript incremented i first.
j = i++; //j is 5, i is 6. Siebel eScript incremented i last.
j = --i; //j is 5, i is 5. Siebel eScript incremented i first.
j = i--; //j is 5, i is 4 Siebel eScript incremented i last.
i++; //i is 5. Siebel eScript incremented i.
```

Using Less Code to Write an Else Statement

The *conditional operator* is a type of operator that allows you to use less code when you write an Else statement. A statement that includes a conditional operator is more difficult to read than an If statement. It is recommended that you use a conditional operator only if the expressions in the If statements are brief.

The following format illustrates how the question mark (?) represents the conditional operator:

```
variable = expressionA ? expressionC : expressionC
```

where:

- *expressionA* is the expression that Siebel eScript evaluates first.
- *expressionB* is the expression that Siebel eScript evaluates if expressionA is true. If expressionA is true, then Siebel eScript replaces the value of the entire expression with the value of expressionB.
- *expressionC* is the expression that Siebel eScript evaluates if expressionA is true. If expressionA is false, then Siebel eScript replaces the value of the entire expression with the value of expressionC.

To use less code to write an Else statement

- Use a conditional operator instead of an Else statement.

Examples of Using the Conditional Operator

In the following example, the expression is true and Siebel eScript sets the value of variableA to 100:

```
variableA = ( 5 < 6 ) ? 100 : 200;
```

Consider the following example:

```
TheApplication().RaiseErrorText("Name is " + ((null==name) ? "unknown" : name));
```

If the name variable contains:

- A null value, then Siebel CRM displays the following text:

  ```
  Name is unknown
  ```

- A value that is not null, such as Pat, then Siebel CRM displays the following text:

  ```
  Name is Pat
  ```

Concatenating Strings

Concatenating is the act of stringing two items together in consecutive order. You can write code that concatenates two or more strings in Siebel eScript.

To concatenate strings

- Use the plus (+) operator between two strings.

Examples of Concatenating Strings

The following example uses the addition (+) operator between two strings:

```
var proverb = "A rolling stone " + "gathers no moss.";
```

This example sets the value of the proverb variable to the following text:

```
A rolling stone gathers no moss.
```

The following example concatenates a string and a number:

```
var newstring = 4 + "get it";
```

This example sets the value of the new string variable to the following text:

```
4get it
```

Using a Bit Operator

Siebel eScript includes operators that you can use to work directly on the bits that reside in a byte or in an integer. To use a bit operator, you must possess knowledge about bits, bytes, integers, binary numbers, and hexadecimal numbers. In most situations you do not need to use a bit operator.

Table 14 describes the bit operators you can use in Siebel eScript.

Table 14. Bit Operators You Can Use in Siebel eScript

Operator	Description	Example
<<	Shift left.	i = i << 2
<<=	Equal shift left.	i <<= 2
>>	Signed shift right.	i = i >> 2
>>=	Equal signed shift right.	i >>= 2
>>>	Unsigned shift right.	i = i >>> 2
>>>=	Equal unsigned shift right.	i >>>= 2
&	Bitwise and.	i = i & 1
&=	Equal bitwise and.	i &= 1
|	Bitwise or.	i = i | 1
|=	Equal bitwise or.	i |= 1
^	Bitwise xor, exclusive or.	i = i ^ 1
^=	Equal bitwise xor, exclusive or.	i ^= 1
~	Bitwise not, complement.	i = ~i

Coding with Siebel eScript

This topic describes how to do some basic tasks that include Siebel eScript. It includes the following topics:

- "Releasing an Object from Memory" on page 50
- "Monitoring the Performance of Your Script" on page 51

Using Script Libraries

The ST eScript engine provides business service script libraries that assist you with developing components that are reusable and modular, which simplifies upgrades and maintenances. You can use script libraries to call global scripts. Script libraries provide the following capabilities:

- Allows you to write code that calls a business service function directly from anywhere in the scripting interface after you declare the business service. You are not required to write code that declares property sets or issue InvokeMethod calls.
- Allows you to write strongly typed methods for predefined business services. You can then use the Script Assist utility to write code that calls these business services. For more information, see "Using Strongly Typed and Typeless Variables" on page 43.

Using script libraries is optional. Siebel CRM supports all code written prior to Siebel 8.0.

Example of Calling a Business Service Function

The following example calls a method directly on the Data Transfer Service without declaring a property set. Calling a business service method directly results in scripts that are shorter and more readable:

```
var oBS : Service = TheApplication ().GetService ("Data Transfer Service");
oBS.SendData ("Name", "John Doe");
```

Example of a Creating Custom Method for a Business Service

You can write a custom method for a business service and make it available in Script Assist. The following example creates SendData, which is a custom wrapper method that resides on the Data Transfer Service:

```
function SendData (sTag : String, sValue : String)
{
  var oPS1 = TheApplication ().NewPropertySet ();
  var oPS2 = TheApplication ().NewPropertySet ();

  oPS1.SetProperty ("Tag", sTag);
  oPS1.SetProperty ("Value", sValue);

  this.InvokeMethod ("SendData", oPS1, oPS2)
}
```

You can write code that intercepts and modifies the calls to the Data Transfer Service in a central location in the SendData method.

Displaying a Custom Method in Script Assist

This topic describes how to display a custom method in Script Assist. For more information, see "About the Script Assist Utility" on page 18. For more information about setting an object property or about using the Server Script Editor to create, save, or compile a script, see *Using Siebel Tools*.

To display a custom method in Script Assist

1. Make a custom method available to the script libraries so that you can call it from Script Assist:
   a. Save the business service method script.
   b. Make sure the script does not contain compile errors.

      If a script library calls a function, then the compiler determines if argument types are valid and do not contain incompatibilities.
   c. In Siebel Tools, make sure the External Use property contains a check mark for the business service object.
2. To access Script Assist from the script editor, press CTRL + SPACE.
3. In your script, enter the name of a business service object followed by a period (.).

   Script Assist displays the default and custom scripted methods that are available for the business service object.
4. Choose the method you must add to your script.

Using Strongly Typed and Typeless Variables

A variable can include one of the following:

- **Strongly typed.** You specify the data type when you declare the variable. ST eScript code supports strong typing. Siebel CRM binds strong typing when you compile the code.
- **Typeless.** Siebel CRM determines the data type at run time. ST eScript code and T eScript code supports typeless variables.

A strongly typed variable typically improves performance over a typeless variable.

You can write code that strongly types all of the primitive data types and object data types. For more information, see "About Primitive Data Types" on page 20 and "About Composite Data Types" on page 20.

Creating a Strongly Typed Variable

This topic describes how to create a strongly typed variable.

To create a strongly typed variable

1. Make sure Siebel Tools uses the ST eScript engine.

   For more information, see *Using Siebel Tools*.

2 When you declare the variable, make sure you add a colon (:) at the end of the variable name.

For example:

```
var VariableA:Date = new Date ();
var VariableB:BusObject;
var VariableC:BusComp;
```

Creating a Typeless Variable

This topic describes how to create a typeless variable.

To create a typeless variable

- Do not specify the data type when you declare the variable.

For example:

```
var VariableA = 0;
var VariableB = new Date ();
var VariableC = new BusObject;
```

In this example, Siebel eScript sets the following types:

- Sets VariableA as an integer
- Sets VariableB as a date
- Types VariableC as a business object

The data type that Siebel CRM sets at run time persists until a subsequent operation causes the interpreter to modify the type again.

Declaring and Using Variables

A *variable* is an object that stores and represents information in a script. Siebel eScript can modify the value of a variable but it cannot modify the value of a literal. For example, to display a name literally, you must use the following code multiple times:

```
TheApplication().RaiseErrorText("Aloysius Gloucestershire Merkowitzky");
```

To simplify this code, the following code uses a variable:

```
var Name = "Aloysius Gloucestershire Merkowitzy";
TheApplication().RaiseErrorText(Name);
```

The value of the Name variable changes, which allows you to use shorter lines of code and to reuse the same lines of code.

About Local and Global Variables

Siebel eScript includes the following types of variables:

- **Local.** A variable that you declare in a function. You can write code that references a local variable only in the function where you declare the variable.
- **Global.** A variable that you declare in one of the following ways:
  - Declare the variable outside of a function.
  - Declare the variable in the general declarations section of the application object.

  You can write code that references or modify a global variable from the following items:
  - Any function that is associated with the Siebel object for which you declare the variable.
  - Any object in a Siebel application where you declare the variable.
  - Another Siebel application.
  - If you declare a global variable outside of a function, then you can reference it from any object that resides in the Siebel application where you declare this variable. For more information, see "Declaring a Global Variable Outside of a Function" on page 45.

If you declare a local variable that uses the same name as a global variable, then you cannot reference this global variable from the function where you declare this local variable.

Siebel VB includes a Global statement. You cannot use this statement in Siebel eScript.

Declaring a Global Variable Outside of a Function

You can write code that declares a variable in a location other than in the declaration section. For example:

```
var global1 = 6;

function ABC()

{

   global1 = 8;

  global2 = 6;

}

var global2 = 8;
```

Using a Local Variable Is Preferable to Using a Global Variable

It is recommended that you use a local variable where possible instead of a global variable for the following reasons:

- A local variable helps you create modular code that is easier to debug and modify.
- A local variable requires fewer resources.
- It is easier for other developers to understand how you use a local variable in a single function than it is to understand how you use a global variable across an entire Siebel application.

- If a subsequent development team encounters an object that you script with a global variable, then this team might not understand the use of the global variable. If the team uses this variable, then the team might introduce defects.
- The scope of a global variable is too large to meet the business requirement and often results in a variable whose lifecycle is not clear.

Instead of using a global variable, it is recommended that you configure Siebel CRM to pass an object as a parameter to a function so that you can control the scope of the parameter. If you are considering using a global variable, then you must consider this usage carefully. If you use a global variable, then do so only rarely and document it thoroughly.

Example of Declaring Local and Global Variables

The following example includes local and global variables:

```
var globalVariable = 1;
function Function1()
{
   var localVariable1 = 1;
   var localVariable2 = 3;
   Function2(d);
}

function Function2(e)
{
   var localVariable3 = 2
   ...
}
```

This example illustrates the following concepts:

- The globalVariable variable is global to the object where you declare it because it is declared outside of a function. Typically you declare all global variables in a general declarations section.
- To create a local variable, you declare it in a function. The following variables are local because this example declares them in a function:
  - localVariable1
  - localVariable2
  - localVariable3
- This example cannot use localVariable3 in Function1 because it is not defined in this function.
- This example uses the d variable in Function1. It uses the e parameter to pass the d variable to Function2.

The following code includes variables that are available to Function1 and Function2:

```
Function1(): globalVariable, localVariable1, localVariable2
Function2(): globalVariable, localVariable3, e
```

Declaring a Variable

This topic describes how to declare a variable.

To declare a variable

- Use the var keyword.

  For example:

  ```
  var perfectNumber;
  ```

You can write code that saves a value in a variable when you declare it. For example:

```
var perfectNumber = 28;
```

Declaring a Variable In a Statement Block

If you declare a variable in a statement block in a method, then you can reference this variable anywhere in the method, including from a statement block that resides in the method where you did not declare the variable.

Determining the Data Type of a Variable

You can use the typeof operator to determine and set the data type of a variable.

To determine the data type of a variable

- Use one of the following formats:
  - var result = typeof variable
  - var result = typeof(variable)

  To improve readability, you can place parentheses around the variable operand, which makes typeof look like the name of a function rather than an operator keyword. Using these parentheses is functionally the same as not using them. They have no impact on program execution.

Immediately after Siebel CRM encounters one of these code lines, it sets the contents of the variable to one of the following string values:

- boolean
- buffer
- function
- object
- number
- string
- undefined

Passing a Value to a Function

This topic describes how to write code that passes a value to a subroutine or a function through a variable or through a reference.

Passing a Value Through a Variable

Siebel eScript can pass a value to a function through a variable. This variable retains the value that it contained before Siebel eScript passes it even though the subroutine or function might modify the passed value. The following example includes this configuration:

```
var VariableA = 5;
var VariableB = ReturnValue(VariableA);

function ReturnValue(VariableC)
{
   VariableC = 2 * VariableC;
   return VariableC ;
}
```

The following occurs in this example:

- VariableA equals 5 and VariableB equals 10.
- VariableC contains a value only while the ReturnValue function runs.
- VariableC does not contain a value after the ReturnValue function finishes.
- Siebel eScript passes VariableA as a parameter to the ReturnValue function and it manipulates this value as VariableC.
- VariableA retains the value that it contained before Siebel eScript passed it.

Passing a Value Through a Reference

Siebel eScript can pass a variable to a subroutine or a function through a reference. However, you use a variable to pass a value for most methods. Each method determines if it can receive a value from a variable or a reference.

A subroutine or function can modify the value. The following example includes this configuration:

```
var VariableA = new Object;
VariableA.name = "Joe";
VariableA.old = ReturnName(VariableA)

function ReturnName(VariableB)
{
   var VariableC = VariableB.name;
   VariableB.name = “Vijay”;
   return VariableC
}
```

The following occurs in this example:

- Siebel eScript passes VariableA to the ReturnName function through a reference.

- VariableB receives a reference to the object, but it does not receive a copy of this object.
- VariableB can reference every property and method of VariableA.
- The ReturnName function modifies the value in VariableB.name to Vijay. The value in VariableA.name also becomes Vijay.
- The Return statement passes a value back to the function that calls it. Siebel CRM does not run any code in a function that follows the Return statement. For more information, see "Return Statement of a Function Object" on page 178.

Preventing a Floating-Point Error

CAUTION: Saving a floating-point number in a variable might cause a loss in precision due to a memory limitation for decimal-to-binary conversion.

Siebel CRM can store a decimal number that does not convert to a finite binary representation with only a small precision error. For example, the following statement might result in Siebel CRM storing VariableA as 142871.450000000001:

```
var VariableA = 142871.45
```

A small precision error will likely have little effect on the precision of a subsequent calculation, depending on the context where you use the number.

A number might be too large for the field that Siebel CRM uses to display it, resulting in an error message that is similar to the following:

```
Value too long for field %1 (maximum size %2)
```

To prevent a floating-point error

- Use the Convert Number to Fixed Decimal Method method at an appropriate location in the calculation or when you save the value in a variable.

  For example, use x.toFixed(2) in a calculation instead of using VariableA. For more information, see "Convert Number to Fixed Decimal Method" on page 159.

Using the Literal Value of a Special Character

Each of the following characters possesses a special programmatic meaning in Siebel eScript:

- Double quotes (")
- Single quote (')
- Semi-colon (;)
- Ampersand (&)
- Hard return

In some situations, you might need to use the literal value of one of these characters. For example:

- Display quotation marks around a phrase in the Siebel client.
- Add a carriage return in a text file.
- Specify a file system path.

To use the literal value of a special character

- Precede the special character with two backslashes (\\).

  You must use two backslashes in Siebel eScript. It recognizes a single backslash as indicating that the next character identifies a character combination. For more information, see "How Siebel eScript Handles Special Characters In a String" on page 88.

Running Browser Script When Siebel CRM Starts a Siebel Application

You can configure Siebel CRM to run Browser Script when it starts a Siebel application. Siebel CRM normally runs code in the declaration section of the Browser Script for a business service when it starts this application. It interprets any code that exists in the general declaration section as HTML. When it loads this application in the Browser, it attaches each Browser script as a Script tag in an HTML page. This configuration allows you to use the general declaration section as the Browser counterpart of the Application Start Server event.

To run Browser script when Siebel CRM starts a Siebel application

- Use code in the general declaration section of the Browser script.

Releasing an Object from Memory

You must explicitly release from memory the following object types when your code no longer requires them:

- Application
- Business component
- Business object
- Configuration item
- CTI data
- CTI service
- Property set
- Web applet
- Web service

This situation is true for T eScript and ST eScript for Browser script and for Siebel VB.

To release an object from memory

- Set the object as a Null object.

  For more information, see "Null Data Type" on page 22.

Monitoring the Performance of Your Script

Starting with Siebel CRM version 8.1, the ST eScript engine allows you to monitor the performance of your script in Siebel CRM. You can identify parts of a script that consumes the most time to process, and then modify it to make it more efficient. Siebel Tools displays profile information in the Tools Script Performance Profiler window. You can export this information to a text file. For more information, see *Using Siebel Tools*.

Guidelines for Using Siebel eScript

This topic describes guidelines for using Siebel eScript. It includes the following topics:

This topic describes only some of the guidelines for using Siebel eScript. For more guidelines, see the topic about guidelines for using Siebel VB and Siebel eScript in *Siebel Object Interfaces Reference*.

Make Sure You Use the Correct Format for Names

A variable name or a function name must include only the following characters:

- **Uppercase ASCII letters.** For example, ABCDEFGHIJKLMNOPQRSTUVWXYZ.
- **Lowercase ASCII letters.** For example, abcdefghijklmnopqrstuvwxyz.
- **Digits.** For example, 0123456789.
- **Underscore (_).**
- **Dollar sign ($).**

A variable name or a function name must use the following format:

- Must begin with a letter, an underscore (_), or a dollar sign ($).

- Cannot include any special characters. For more information, see "Special Characters" on page 52.
- Cannot include white space. Siebel eScript uses white space to separate names. For more information, see "Use White Space to Improve Readability" on page 56.
- Cannot include a reserved word. For more information, see "Reserved Words" on page 53.
- Can include any length.

The following example names are valid:

```
George
Martha7436
annualReport
George_and_Martha_prepared_the_annualReport
$alice
CalculateTotal()
$SubtractLess()
_Divide$All()
```

The following example names are not valid:

```
1george
2nancy
this&that
Martha and Nancy
What?
=Total()
(Minus)()
Add Both Figures()
```

Special Characters

Table 15 lists the characters that Siebel eScript recognizes as special characters.

Table 15. Special Characters

Special Character	Description
<	Less than symbol.
>	Greater than symbol.
&	Ampersand symbol.
|	Pipe symbol.
=	Equal to sign.
!	Exclamation point.
*	Asterisk.
/	Forward slash.
%	Percentage symbol.
^	Caret symbol.

Table 15. Special Characters

Special Character	Description
~	Tilde symbol.
?	Question mark.
:	Colon.
{	Open curly bracket.
}	Close curly bracket.
;	Semi-colon.
(	Open parenthesis.
)	Close parenthesis.
[	Open square bracket.
]	Close square bracket.
.	Period.
‘	Single quote.
"	Double quote.
'	Apostrophe.
#	Pound symbol.

Reserved Words

The following words have special meaning in Siebel eScript. You cannot write code that uses any of them as a variable name or a function name:

break	export	super
case	extends	switch
catch	false	this
class	finally	throw
const	for	true
continue	function	try
debugger	if	typeof
default	import	while
delete	in	with
do	new	var
else	null	void
enum	return	

Make Sure You Use the Correct Case

Siebel eScript is case-sensitive. For example, the testvar variable is different from the TestVar variable. Each of these variables can exist in a script at the same time. The following example defines two different variables:

```
var testvar = 5;
var TestVar = "five";
```

The name of a method or function in Siebel eScript is case-sensitive. For example, the following code creates an error on the Siebel Server:

```
TheApplication().RaiseErrorText("an error has occurred");
```

The following example creates an error in a Siebel application:

```
TheApplication().raiseerrortext("an error has occurred");
```

A control statement is case-sensitive. For example, the following statement is valid:

```
while
```

The following statement is not valid:

```
While
```

Use Expressions, Statements, and Statement Blocks

An *expression* includes two or more terms that perform a mathematical or logical operation. These terms are typically variables or functions that you can use with an operator to produce a string or numeric result. You can write code that uses an expression to configure Siebel eScript to do the following work:

- Perform a calculation.
- Manipulate a variable.
- Concatenate a string.

The following example statement includes an expression. It computes a sum and saves it in a variable:

```
var TestSum = 4 + 3
```

Note the following:

- Siebel CRM runs Siebel eScript code one statement at a time from the beginning of the code to end of the code.
- You can use a semicolon at the end of a statement, although Siebel eScript does not require this format.
- To make your script easier to read and edit, it is recommended that you write each statement on a separate line, with or without a semicolon.

- A *statement block* is a group of statements that Siebel eScript treats as one statement. You use curly brackets ({}) to enclose a statement block. To simplify reading, it is recommended that you indent the statements in a statement block.

Running Statements In a Loop

A *While statement* is a type of statement that causes Siebel eScript to run the statement that occurs immediately after the While statement in a loop. If you enclose multiple statements in curly brackets, then Siebel eScript treats them as one statement and runs them in the loop. The following example includes this usage:

```
while( ThereAreUncalledNamesOnTheList() == true)
{
   var name = GetNameFromTheList();
   CallthePerson(name);
   LeaveTheMessage();
}
```

Siebel eScript treats the three lines that occur after the While statement as one unit. The brackets cause Siebel eScript to run the script through each line until it calls every name that resides in the list. If you remove these brackets, then it does the following:

- Runs the loop only for the first line.
- Processes the names on the list but only calls the last name.

Use a Primitive Data Type Instead of an Object Data Type

It is recommended that you use an object only if you must use a property that is specific to this object type. If an equivalent primitive data type exists, then use the primitive. A primitive data type provides superior performance. An object data type consumes more resources than a primitive data type.

Table 16 lists primitive data types that are equivalent to object data types. For example, if you do not need to use a string-specific object or conversion method, then use the chars primitive instead of a String object.

Table 16. Primitive Data Types That Are Equivalent to Object Data Types

Primitive Data Type	Object Data Type
Chars	String
Float	Number
Bool	Boolean

Use White Space to Improve Readability

A *white-space character* is a type of character that determines the spacing and placement of text in your code. Each of the following items is an example of a white-space character:

- Space
- Tab
- Carriage-return
- New line

White space makes your code easier to read. Siebel eScript ignores white space characters.

A line of script ends with a carriage-return character. Each line is typically a separate statement. In some editors a line ends with a carriage-return and the following linefeed pair:

```
\r\n
```

Siebel eScript typically interprets as white space one or more white-space characters that exist between names of methods and functions. Each of the following Siebel eScript statements are equivalent to one another:

```
var x=a+b
var x = a + b
var x =     a     +     b
var x = a+
        b
```

White space separates the names, methods, functions, and variables. For example, ab is one variable name, and a b are two variable names. The following example is valid:

```
var ab = 2
```

The following example is not valid:

```
var a b = 2
```

Some developers use spaces and not tabs because tab size settings vary from editor to editor. If a developer uses only spaces, then the script format is consistent across editors.

Using White Space in a String Literal Can Cause Errors

CAUTION: Siebel eScript treats white space in a string literal differently from how it treats white space that occurs elsewhere. Placing a line break in a string causes Siebel eScript to treat each line as a separate statement. Each of these statements contains an error because they are not complete. To avoid this situation, you must keep string literals on a single line or create separate strings, and then use the string concatenation operator to concatenate them.

For example:

```
var Gettysburg = "Fourscore and seven years ago, " +
"our fathers brought forth on this continent a " +
"new nation.";
```

For more information, see “Concatenating Strings” on page 40.

Use Comments to Document Your Code

A *comment* is text in a script that you can use to document the script. It can describe the intent of the code flow, which simplifies modifications and debugging. Siebel eScript skips comments. Siebel eScript includes the following types of comments:

- **End-of-line comment.** Begins with two forward slashes (//). It ignores any text that occurs from after these slashes to the end of the current line. It begins interpreting the code that occurs on the next line.
- **Statement block comment.** Begins with a forward slash and an asterisk (/*). Ends with an asterisk and a forward slash (*/). Text between these markers is a comment even if the comment extends over multiple lines. You cannot write code that includes a statement block comment in a statement block comment. You can include an end-of-line comment in a statement block comment.

The following code includes valid comments:

```
// this is an end of line comment

/* this is a statement block comment.
This is one big comment block.
// this comment is okay in the statement block.
The interpreter ignores it.
*/

var FavoriteAnimal = "dog"; // except for poodles

//This line is a comment but
var TestStr = "This line is not a comment.";
```

Make Sure the JavaScript Interpreter Can Run a Function

If a function is unique to Siebel eScript, then you must make sure that the JavaScript interpreter that runs the script supports Siebel eScript functions. Avoid using a function that is unique to Siebel eScript in a script that Siebel CRM might use with a JavaScript interpreter that does not support the function.

4 Statements Reference

This chapter describes reference information for statements you can use in Siebel eScript. It includes the following topics:

Break Statement

The Break statement does the following:

- Stops the innermost loop of the following statements:
  - For
  - While
  - Do
- Controls the flow in a Switch statement.

Format A

```
break;
```

Format B

```
break label;
```

Table 17 describes the arguments you can use with the Break statement.

Table 17. Arguments for the Break Statement

Argument	Description
label	The name of the label that indicates where this statement must resume running the script. This label includes the name of a method or a function followed by a colon.

Usage

You can write code that uses the Break statement only in the following situations:

- **In a loop.** Stops the loop if the code no longer requires the loop.
- **In a Switch statement.** Stops Siebel eScript from running any code that occurs after the Label statement. Causes Siebel eScript to exit the Switch statement.

If you use the Break statement in a nested loop, then it causes Siebel eScript to stop running the script only in this nested loop. If the Break statement occurs in a nested loop, then you can use the label argument to indicate the beginning of the loop that Siebel eScript must stop.

Example

For an example, see "Switch Statement" on page 67.

Continue Statement

The Continue statement starts a new iteration of a loop. It ends the current iteration of a loop, and then begins the next loop. Siebel eScript evaluates any conditional expressions before the loop reiterates.

Format A

```
continue;
```

Format B

```
continue label;
```

Table 18 describes the argument.

Table 18. Arguments for the Continue Statement

Argument	Description
label	The name of the label that indicates where to resume running the code. This label includes the name of a method or a function followed by a colon.

Example

The following example writes the numbers 1 through 6 and 8 through 10, and then the following string:

```
.Test
```

The use of the Continue statement after the if (i==7) statement prevents Siebel eScript from running the loop on the seventh iteration, but keeps running the loop:

```
var i = 0;
while (i < 10)
{
   i++;
   if (i==7)
      continue;
   document.write(i + ".Test");
}
```

Do While Statement

The Do While statement processes the code that the statement_block argument identifies repeatedly until the statement meets the value that the condition argument contains. The condition argument occurs at the end of the loop. Siebel eScript tests the condition only after the loop runs. A Do While loop always runs at least one time before Siebel eScript examines the condition.

Format

```
do
{
   statement_block;
} while (condition)
```

Table 19 describes the arguments you can use with the Do While statement.

Table 19. Arguments for the Do While Statement

Argument	Description
statement_block	One or more statements that Siebel eScript runs in the loop.
condition	An expression that describes the condition that Siebel eScript uses to repeat the loop.

Example

The following example increments a value and prints the new value to the screen until the value reaches 100:

```
var value = 0;
do
{
```

```
      value++;
      Clib.sprintf(value);
   } while( value < 100 );
```

For Statement

The For statement repeats a series of statements a fixed number of times. Siebel eScript does the following when it runs the For statement:

1. Evaluates the following expression:

   ```
   counter = start
   ```

2. Does one of the following:

   - **Condition is true or no conditional expression exists.** It does the following work:
     - Runs the For statement.
     - Increments the counter.
     - Goes to Step 1.
   - **Condition is false.** It does the following work:
     - Exits the For statement.
     - Runs the code line that occurs immediately after the For statement.

Format

```
for ( [var] counter = start; condition; increment )
{
   statement_block;
}
```

Table 20 describes the arguments for the For statement.

Table 20. Arguments for the For Statement

Argument	Description
counter	A numeric variable for the loop counter.
start	The initial value of the counter.

Usage

If the counter argument is not declared, then you must use the Var statement to declare it. Although it is declared in the For statement, the scope of the counter variable is local to the entire function that includes the for loop.

If you use multiple counters, then you must use a comma to separate each counter. For example:

```
for (var i = 1, var j = 3; i < 10; i++, j++)
   var result = i * j;
```

If you configure Siebel CRM to modify the value in the counter argument other than through the increment that occurs as a result of running the For statement, then your script might be difficult to maintain or debug.

Example

For an example of the For statement, see "Evaluate Expression Method" on page 173.

For In Statement

The For In statement loops through the properties of an associative array or object. You cannot use it with a nonassociative array. For more information, see "About Associative Arrays" on page 78.

You cannot write code that references a property that is marked with the DONT_ENUM attribute in a For In statement. The DONT_ENUM attribute is a predefined attribute that you cannot modify.

Format

```
for (LoopVar in object)
{
   statement_block;
}
```

Table 21 describes the arguments for the For In statement. The statement block runs one time for every element in the associative array or property of the object.

Table 21. Arguments for the For In Statement

Argument	Description
object	An associative array or object: ■ The object must possess at least one defined property. ■ The associative array must possess at least one defined element.
LoopVar	A variable that iterates over every element that resides in the associative array or property of the object. For each iteration of the loop, the LoopVar argument identifies the name of a property of the object or an element of the array. You can write code that references it with a statement that uses the following format: ■ *object*[*LoopVar*] ■ *array_name*[*LoopVar*]

Example

The following example creates an object named obj, and then uses the For In statement to read the object properties:

```
function PropBtn_Click ()
{
   var obj = new Object;
   var propName;
   var msgtext = "";

   obj.number = 32767;
   obj.string = "Welcome to my world";
   obj.date = "April 25, 1945";

   for (propName in obj)
   {
      msgtext = msgtext + "The value of obj." + propName +
         " is " + obj[propName] + ".\n";
   }
   TheApplication().RaiseErrorText(msgtext);
}
```

Running this code produces the following results:

```
The value of obj.number is 32767.
The value of obj.string is Welcome to my world.
The value of obj.date is April 25, 1945.
```

For an example of the For In statement used with an associative array, see "About Associative Arrays" on page 78.

Goto Statement

The Goto statement causes Siebel eScript to go to a specific point in a function. You can write code that directs Siebel eScript to go to any location in a function. It is recommended that you use a Goto statement only rarely because it makes it difficult to follow the code flow.

Format

goto *label*;

Table 22 describes the argument for the Goto statement.

Table 22. Argument for the Goto Statement

Argument	Description
label	Indicates the code line where Siebel eScript must resume running the code. You must place a label argument at the point where Siebel eScript must resume running the code.

Example

The following example uses a label argument to loop continuously until the number is greater than 0:

```
function clickme_Click ()
{
restart:
   var number = 10;
   if (number <= 0 )
      goto restart;
   var factorial = 1;
   for ( var x = number; x >= 2; x-- )
      factorial = (factorial * x);
   TheApplication().RaiseErrorText( "The factorial of " +
      number + " is " + factorial + "." );
}
```

If Statement

The If statement tests a condition and proceeds depending on the result.

Format A

```
if (condition)
   statement;
```

Format B

```
if (condition)
{
   statement_block;
}
[else if (condition)
{
   statement_block;
}]
[else
{
   statement_block;
}]
```

Table 23 describes the arguments for the If statement.

Table 23. Arguments for the If Statement

Argument	Description
condition	An expression that evaluates to true or false.
statement_block	If the expression is: ■ **True.** Siebel eScript runs the statement or statement_block. ■ **False.** Siebel eScript skips the statement or statement_block.

Usage

If you require multiple statements, then use Format B.

The following example includes an If statement:

```
if ( i < 10 )
{
   TheApplication().RaiseErrorText("i is less than 10.");
}
```

If Siebel eScript runs only a single If statement, then the curly brackets are not required. You can use them to clarify your code.

Else Clause

The else clause is an extension of the If statement. It allows you to run code if the condition in the If statement is false. The following example includes the else clause:

```
if ( i < 10 )
   TheApplication().RaiseErrorText("i is less than 10.");
else
   TheApplication().RaiseErrorText("i is not less than 10.");
```

Using More Than One If Statement

If you require more than one If statement, then you must use curly brackets to group the statements. For example:

```
if ( i < 10 )
{
   i += 10;
   TheApplication().RaiseErrorText ("Original i was less than 10, and has now been
   incremented by 10.");
}
else
{
   i -= 5;
```

```
      TheApplication().RaiseErrorText ("Original i was at least 10, and has now been
      decremented by 5.");
   }
```

This example includes an else clause in an If statement. This If statement tests for multiple conditions.

Example

The following example includes an else clause:

```
   if ( i < 10 )
   {
         TheApplication().RaiseErrorText("i is less than 10.")
   }
      else if ( i > 10 )
   {
         TheApplication().RaiseErrorText("i is greater than 10.");
   }
   else
   {
      TheApplication().RaiseErrorText("i is 10.");
   }
```

For another example, see "Set Time Method" on page 137.

For more information, see "Switch Statement" on page 67.

Switch Statement

The Switch statement makes a decision according to the value of a variable or expression. It chooses among alternatives when each choice depends on the value of a single variable. Siebel eScript does the following:

1. Evaluates the switch_variable argument.
2. Compares the values in the Case statements, and then does one of the following depending on if it finds a match:
   - **Finds a match.** Runs the statement block that follows the Case statement whose value matches the value in the switch_variable argument. It runs until it reaches the end of the statement block or until a Break statement causes it to exit the statement block.
   - **Does not find a match.** If a default statement exists, then it runs the default statement.

Usage

Make sure you end a Case statement with a Break statement.

Format

```
switch( switch_variable )
{
   case value1:
      statement_block
      break;
   case value2:
      statement_block
      break;
   .
   .
   .
   [default:
      statement_block;]
}
```

Table 24 describes the arguments for the Switch statement.

Table 24. Arguments for the Switch Statement

Argument	Description
switch_variable	The argument on whose value the course of action depends.
valuen	Values of the switch_variable argument, which are followed by a colon.
statement_block	One or more statements that Siebel eScript runs if the value of switch_variable argument is the value in the Case statement.

Example

This example configures Siebel CRM to perform an action depending on an account type. In this example, a letter indicates the type of account:

```
switch ( key[0] )
{
case 'A':
   I=I+1;
   break;
case 'B':;
   I=I+2
   break;
case 'C':
   I=I+3;
   break;
default:
   I=I+4;
   break;
}
```

Siebel eScript runs code in the statement block until it encounters a Break statement. In this example, if you remove the Break statement that occurs after the I=I+2 statement, then Siebel eScript runs the following code:

- I=I+2
- I=I+3

For more information, see "If Statement" on page 65.

Throw Statement

The Throw statement causes Siebel eScript to stop running code if an error occurs.

Format

```
throw exception
```

Table 25 describes arguments for the Throw statement.

Table 25. Arguments for the Throw Statement

Argument	Description
exception	An object in an error class.

Usage

In the following example, the Throw statement stops the script after Siebel CRM displays an error message:

```
try
{
   do_something;
}
catch( e )
{
   TheApplication().Trace (e.toString()));

throw e;
}
```

Using the Throw Statement with Nested Try Catch Blocks

If any error occurs while processing a statement in a try block, then Siebel eScript creates an exception. An outer catch block can handle this exception. For example, assume a section of code includes three levels of try catch blocks:

1. The innermost catch block includes a throw statement. An exception occurs.
2. The catch statement in the level two block catches this exception.
3. The catch statement in the level two block throws this exception to the level one block.
4. The catch block at level one handles this exception.

The following code illustrates this example:

```
try
   {
   do_something;
   try
      {
      do_something;
      }
         catch(e)
      {
      TheApplication().Trace(e.toString());
      throw e;
      }
    }
      catch(e)
   {
   TheApplication().RaiseErrorText("Error Occurred "+e.toString());
}
```

Avoiding an Exception Error That Is Not Handled

You can write code that uses the RaiseErrorText method or the RaiseError method instead of the Throw statement to avoid receiving an unhandled exception error in the text that the Get Buffer Data method returns. If the Siebel Run-Time Engine creates an error message, or if the Throw statement creates an error message, then Siebel CRM adds the following text to the error message:

```
Unhandled Exception
```

Siebel CRM does this to distinguish an error message that the RaiseErrorText method or that the RaiseError method creates from an error that the Siebel Run-Time Engine creates or that the Throw statement creates.

For more information, see “Get Buffer Data Method” on page 113 and “Try Statement” on page 70.

Try Statement

The Try statement processes an exception. It handles functions that can raise an *exception*, which is an error condition that causes the script to branch to another routine. It can include the following clauses:

- **Catch clause.** Handles the exception. To raise an exception, you use the Throw statement. For more information, see “Throw Statement” on page 69.
- **Finally clause.** Performs cleanup work. For example, removing object references.

You can write code that does the following to trap errors that a statement block creates:

- Place code that must trap errors in a Try statement.
- Follow the Try statement with a catch clause. You can write code that uses the exception_handling_block argument in this catch clause to process the exception.

Format

```
try
{
   statement_block
}
catch
{
   exception_handling_block
   [throw exception]
}
finally
{
   statement_block_2
}
```

Table 26 describes the arguments for the Try statement.

Table 26. Arguments for the Try Statement

Argument	Description
statement_block	A statement block that can create an error.
exception_handling_block	A statement block that processes the error.
exception	An error of a named type.
statement_block_2	A statement block that Siebel eScript always runs unless the Try statement transfers control to elsewhere in the script.

Example

The following example demonstrates the format of a Try statement that includes a catch clause. In this example, Siebel eScript continues to run the script after it displays the error message:

```
try
{
   do_something;
}
catch( e )
{
   TheApplication().RaiseErrorText(Clib.rsprintf(
      "Something bad happened: %s\n",e.toString()));
}
```

Example Usage of the Finally Clause

The finally clause includes code that Siebel eScript must run before it exits the Try statement, regardless of if a catch clause stops running the script. You can write code that uses one of the following statements to exit a finally clause:

- Goto
- Throw

- Return

CAUTION: A Return statement in the finally clause suppresses any exceptions that Siebel eScript creates in the method or that it passes to the method. It skips statements in the finally clause only if the finally clause redirects flow to another part of the script.

The following example includes a finally clause. Siebel eScript continues running this code after the no_way statement. It ignores the Return statement:

```
try
{
   return 10;
}
finally
{
   goto no_way;
}

no_way: statement_block
```

While Statement

The While statement runs a section of code repeatedly until an expression evaluates to false. It does the following:

1. Examines the expression.
2. If the expression is true, then it does the following:
   a. Runs the code in the statement block that occurs after the condition argument.
   b. Repeats Step 1.
3. If the expression is false, then Siebel eScript runs the code that occurs immediately after the statement block.

A while loop repeats until the value in the condition argument is false.

Format

```
while (condition)
{
     statement_block;
}
```

Table 27 describes the arguments for the While statement.

Table 27. Arguments for the While Statement

Argument	Description
condition	Includes a value that determines when to stop running the loop. You must enclose this argument in parentheses.
statement_block	One or more statements that Siebel eScript runs while the condition argument is true.

Example

The following example includes a While statement that includes two lines of code in a statement block:

```
while(ThereAreUncalledNamesOnTheList() != false)
{
   var name = GetNameFromTheList();
   SendEmail(name);
}
```

With Statement

The With statement associates a default object with a statement block. It only applies to the code that resides in the statement block where the With statement occurs, regardless of how Siebel eScript enters or exits the statement block. If Siebel eScript exits a With statement, then the With statement no longer applies.

You cannot write code that uses a Goto statement or a label to enter or exit the middle of a statement block that resides in a With statement.

Format

```
with (object)
{
     method1;
     method2;
     .
     .
     .
     methodn;
}
```

Table 28 describes the arguments of the With statement.

Table 28. Arguments of the With Statement

Argument	Description
object	An object where you must use multiple methods.
method1, method2, methodn	Methods that Siebel eScript runs with the object. The With statement prefixes each method with the object name and a period.

Example

The following example includes a With statement:

```
var bcOppty;
var boBusObj;
boBusObj = TheApplication().GetBusObject("Opportunity");
bcOppty = boBusObj.GetBusComp("Opportunity");
var srowid = bcOppty.GetFieldValue("Id");

try
{
   with (bcOppty)
   {
      SetViewMode(SalesRepView);
      ActivateField("Sales Stage");
      SetSearchSpec("Id", srowid);
      ExecuteQuery(ForwardOnly);
   }
}
finally
{
   boBusObj = null;
   bcOppty = null;
}
```

The code in the With statement block is equivalent to the following code:

```
bcOppty.SetViewMode(SalesRepView);
bcOppty.ActivateField("Sales Stage");
bcOppty.SetSearchSpec("Id", srowid);
bcOppty.ExecuteQuery(ForwardOnly);
```

5 Methods Reference

This chapter describes reference information for methods that you can use in Siebel eScript. It includes the following topics:

Overview of Methods Reference

In addition to the methods that this chapter describes, you can also reference the following items in Siebel eScript. For detailed information, see the Siebel eScript quick reference chapter in *Siebel Object Interfaces Reference*:

- Applet object
- Application object
- Business component object
- Business service object
- Property set

Usage of the Term Put

The term *put* means to replace existing data. For example, if you put eight bytes of data to a BLOB object starting at offset 0, then Siebel CRM replaces data that currently resides in bytes 0 through 7 of the BLOB object with the input data. This book uses this definition of put throughout this chapter.

Array Methods

This topic includes the following topics:

Overview of Array Methods

Note the following:

- An *array* is a class of object that holds multiple values instead of one value. To reference a single value in an array, you use an array index number or string that is associated with this value.
- An *array element* is the value of an array object. It can include any data type. Siebel CRM does not require that the elements in an array be the same type, and it does not limit the number of elements that an array can include.
- The *index number* is a number or a string that identifies the array element. This number follows the array name and you place it in square brackets.

The following example statements store values in an array:

```
var array = new Array;
array[0] = "fish";
array[1] = "fowl";
```

```
array["joe"] = new Rectangle(3,4);
array[foo] = "creeping things"
array[goo + 1] = "and so on."
```

Array elements can be noncontiguous. For example, an array can include the following items:

- An element at index 0
- No element at index 1
- An element at index 2

An array typically starts at index 0. It does not typically start at index 1.

Example of Using an Array

An array can use a number as an index, so it allows you to work with sequential data. For example, to keep track of how many jelly beans you eat each day, you can graph your jelly bean consumption at the end of the month. An array provides a solution for storing such data. For example:

```
var April = new Array;
April[1] = 233;
April[2] = 344;
April[3] = 155;
April[4] = 32;
```

In this example, one variable contains all the data. You can write code that examines the value of April[x] to determine how many jelly beans you ate on day x. For example:

```
for(var x = 1; x < 32; x++)
TheApplication().Trace("On April " + x + " I ate " + April[x] +


   " jellybeans.\n");
```

About Array Functions

You use the following operator and an array function to create an array:

new

Table 29 describes different ways that you can use the array function.

Table 29. Example Usage of the Array Function

Example Array Function	Description
`var a = new Array();an`	This code initializes the following variable as an array with no elements: a The parentheses are optional.
`var b = new Array(31);`	This code creates an array that includes 31 array elements. If you must create an array that includes a predefined number of array elements, then you can use the number of elements as a argument of the Array function when you declare the array.
`var c = new Array(5, 4, 3, 2, 1, "blast off");`	This code creates an array that includes six elements: ■ c[0] is set to 5. ■ c[1] is set to 4. ■ And so on up to c[5], which is set to the string "blast off". The first element of the array is c[0]. It is not c[1]. You can write code that passes elements to the Array function, which creates an array that contains the arguments that your code passes.

You can write code that creates an array dynamically. If you write code that uses an index in brackets to reference a variable, then the variable becomes an array. If you use this technique to create an array, then you cannot use the methods and properties with an associative array.

About Associative Arrays

An *associative array* is a type of array that uses a string as an index element. This capability is useful if you must associate a value with a specific name. For example, you can create a month array where the elements are the names of the months and the values are the number of days in the month.

You use a string as an index to reference items in an associative array. For example:

```
array_name["color"] = "red";
array_name["size"] = 15;
```

The associative array is the only type of array that you can use with the following type of statement:

```
for in
```

This statement loops through every element in an associative array or object, regardless of how many elements it contains. For more information, see “For In Statement” on page 63.

Siebel CRM uses a hash table to implement the associative array, so the elements are not in an order that an indexed array uses, and you cannot use array methods with an associative array, such as split, join, or length.

Example of Using an Associative Array

The following example creates an associative array of months and days, and totals the number of days:

```
// open file
var fp = Clib.fopen("c:\\months.log", "at");

// populate associative array
var months = new Array();
months["November"] = 30;
months["December"] = 31;
months["January"] = 31;
months["February"] = 28;

// iterate through array items
var x;
var total = 0;
for (x in months)
   {
    // write array items name and value to file
    Clib.fputs(x + " = " + months[x] + "\n",fp);
    // Add this month's value to the total
    total = total + months[x];
   }
Clib.fputs ("Total = " + total + "\n",fp);

//close file
Clib.fclose(fp);
```

The following is the output from this example:

```
November = 30
December = 31
January = 31
February = 28
Total = 120
```

Add Array Elements Method

The Add Array Elements method adds the elements that you define in the element argument to the end of the array. It adds these elements in the order that you define these arguments.

Format

arrayName.push([*element1*,*element2*, ..., *elementn*])

Table 30 describes the arguments for the Add Array Elements method.

Table 30. Arguments for the Add Array Elements Method

Argument	Description
element1, element2, . . . elementn	A list of elements to add to the array.

Example

The following example includes the Add Array Elements method:

```
var a = new Array(1,2);
TheApplication().RaiseErrorText(a.push(5,6) + "   " + a);
// Displays 4  1,2,5,6, the length and the new array.
```

Concatenate Array Method

The Concatenate Array method concatenates all the elements of an array into a string. It returns a concatenated string that includes individual array element values that are separated by commas. It does not include any input arguments.

Format

```
concat()
toLocaleString()
```

Converting a Concatenated Array to Another Language

The toLocaleString statement works just like the concat statement but it converts the string to another language according to the locale setting.

Example

The following example includes the Concatenate Array method:

```
var v = new Array;
v[0] = 7;
v[1] = 3;
v.concat();   // The result would be "7,3"
```

Create Array Elements Method

The Create Array Elements method creates a string of array elements. It returns a string that contains the array elements. A comma or the separatorString argument separates each element.

Format

arrayName.join([*separatorString*])

Table 31 describes the arguments for the Create Array Elements method.

Table 31. Arguments for the Create Array Elements Method

Argument	Description
separatorString	A string of characters that occur between consecutive elements of the array. If you do not use a separatorString argument, then you can use a comma.

Usage

Commas separate the array elements by default. The following example sets the value that the string variable contains to 3,5,6,3:

```
var a = new Array(3, 5, 6, 3);
var string = a.join();
```

To separate the array elements, you can write code that passes another string as an optional argument to the Create Array Elements method.

Example

The following example creates a string that contains a value of 3*/*5*/*6*/*3:

```
var a = new Array(3, 5, 6, 3);
var string = a.join("*/*");
```

Delete Last Array Element Method

The Delete Last Array Element method does the following work:

1. Gets the length of the current Array object.
2. If the length is defined or is not 0, then it does the following:
   a. Returns the last element.
   b. Deletes the last element.
   c. Decreases the length of the current array object by one.
3. If the length is undefined or is 0, then it returns an undefined value.

The Delete Last Array Element method works on the end of an array. You must use the Array Shift method to work on the beginning of an array.

Format

arrayName.pop()

Example

The following example includes the Delete Last Array Element method:

```
var a = new Array( "four" );
TheApplication().RaiseErrorText("First pop: " + a.pop() + ", Second pop: " +
a.pop());
// First displays the last (and only) element, the string "four".
// Then displays "undefined" because the array is empty after
// the first call removes the only element.
```

Get Largest Array Index Method

The Get Largest Array Index method returns the number of the highest index that the array contains, plus 1. This return value does not necessarily include the actual number of elements in an array because Siebel eScript does not require elements to be contiguous.

Format

arrayName.length

Example

The following example includes two arrays:

```
var ant = new Array;     var bee = new Array;
ant[0] = 3               bee[0] = 88
ant[1] = 4               bee[3] = 99
ant[2] = 5
ant[3] = 6
```

The length property of ant and bee is equal to 4 even though ant includes twice as many array elements as bee. To remove array elements, you can write code that modifies the value of the length property. For example, if you write code that modifies ant.length to 2, then ant loses any elements that occur after the first two elements, and Siebel CRM loses the values that it stored at the other indices. If you set bee.length to 2, then bee includes the following elements:

- bee[0], with a value of 88
- bee[1], with an undefined value

Get Subarray Method

The Get Subarray method gets the array elements that exist in a range starting with the value that the first element argument identifies and ending with the value that the last element argument identifies. It returns a new array.

Format

slice (*first element*, *last element*)

Table 32 describes the arguments for the Get Subarray method.

Table 32. Arguments for the Get Subarray Method

Argument	Description
first element	The first element that this method returns.
last element	The last element minus one that this method returns.

Example

The following example includes the Get Subarray method:

```
var v = new Array;
var u;
v[0] = 7;
v[1] = 3;
v[2] = 4;
v[3] = 5;
u = v.slice ( 1, 3);  // u creates new array containing v[1] and v[2] values. For
example, u[0] = 3, u[1] = 4.
v.shift();  // Now v[0] is 3, v[1] is 4
```

Insert Array Elements Method

The Insert Array Elements method inserts array elements into an array. It returns an array that includes the elements that it removed from the original array. It does the following work:

1. Beginning at the value that the start argument specifies, it deletes the number of array elements that the deleteCount argument specifies.
2. Inserts these deleted elements into the newly created return array in the same order that it uses to delete them.
3. To make room for new elements, it adjusts the elements in the current array object.
4. Inserts the array elements that you specify in the element1, element2, . . . elementn argument. It inserts these elements sequentially in the space that it creates in Step 3.

Format

arrayName.splice(*start*, *deleteCount*[, *element1*, *element2*, . . . *elementn*])

Table 33 describes the arguments for the Insert Array Elements method.

Table 33. Arguments for the Insert Array Elements Method

Argument	Description
start	Identifies the index where this method inserts the new array elements. This method does the following: ■ If start is negative, then it uses the value of the length of the array plus start. It inserts at the position counting back from the end of the array. For example, the following code inserts from the last element in the array: `start = -1` ■ If start is larger than the index of the last element, then it uses the length of the array. It appends new elements to the end of the array.
deleteCount	Identifies the number of array elements to remove from the array. If deleteCount is larger than the number of elements that exist in the array, then this method removes all of the elements.
element1, element2, . . . elementn	A list of elements that this method inserts in the array.

Example

The following example includes the Insert Array Elements method:

```
var a = new Array( 1, 2, 3, 4, 5 );
TheApplication().RaiseErrorText(a.splice(1,3,6,7) + "   " + a);
// Displays 2,3,4   1,6,7,5
// Beginning at element in position 1, three elements (a[1], a[2], a[3] = 2,3,4)
// are replaced with 6,7.
```

Reverse Array Order Method

The Reverse Array Order method reverses the order of the array elements so that the last element becomes the first element. It returns the elements in reverse order. It returns this reverse order in the arrayName argument. It reverses the existing array. It does not return a new array.

Format

arrayName.reverse()

Example

The following example includes the Reverse Array Order method:

```
var communalInsect = new Array;
communalInsect[0] = "ant";
communalInsect[1] = "bee";
communalInsect[2] = "wasp";
communalInsect.reverse();
```

This example produces the following array:

```
communalInsect[0] == "wasp"
communalInsect[1] == "bee"
communalInsect[2] == "ant"
```

Shift Array Left Method

The Shift Array Left method shifts all array elements by one position to the left. The first element is lost. It returns the modified array. It does not include any input arguments.

Format

```
shift()
```

Example

The following example includes the Shift Array Left method:

```
var v = new Array;
v[0] = 7;
v[1] = 3;
v[2] = 4;
v[3] = 11;

v.shift();  // now v[0] becomes 3, v[1] becomes 4, v[2] becomes 11
```

Shift Array Right Method

The Shift Array Right method shifts array elements to the right. Siebel eScript assigns the argument values sequentially starting from the first element in the array. It fills the remaining array elements with values from the original array starting with the first value.

Format

unshift *(integer)*

You can include any number of arguments.

Example

The following example includes the Shift Array Right method:

```
var v = new Array;
v[0] = 7;
v[1] = 3;
v[2] = 4;
v[3] = 5;
v.unshift (11, 12); // v[0] is 11 now, v[1] is 12, v[2] is 7 , v[3] is 3
```

Sort Array Method

The Sort Array method sorts array elements into an order that you specify. It returns the sorted array elements.

Format

arrayName.sort([*compareFunction*])

Table 34 describes the arguments for the Sort Array method.

Table 34. Arguments for the Sort Array Method

Argument	Description
compareFunction	Specifies the sort order. This method does the sort differently depending on if you include the compareFunction argument: ■ **Include the compareFunction argument.** It sorts array elements according to the return value that the compare function contains. ■ **Do not include the compareFunction argument.** It converts array elements to strings before it sorts them. It sorts numbers in ASCII order, comparing them from left to right. For example, 32 comes before 4. The compareFunction argument allows you to modify this sort behavior.

Example

The following example uses the Sort Array method with and without a compare function:

```
function compareNumbers(a, b)
{
   return a - b;
}
var a = new Array(5, 3, 2, 512);
var fp = Clib.fopen("C:\\log\\Trace.log", "a");
Clib.fprintf(fp, "Before sort: " + a.join() + "\n");
a.sort(compareNumbers);
Clib.fprintf(fp, "After sort: " + a.join() + "\n");
Clib.fclose(fp);
```

This example does the following:

1. Displays the results of a sort without the function.
2. Uses the following function to sort the numbers:

   ```
   compareNumbers(a, b)
   ```

   In this function, if a and b are two array elements that Siebel eScript compares, then Siebel eScript does the following:

   - If compareNumbers(a, b) is less than zero, then it gives b a lower index than a.
   - If compareNumbers(a, b) returns zero, then it does not modify the order of a and b.
   - If compareNumbers(a, b) is greater than zero, then it gives b a higher index than a.

String Methods

This topic describes string methods. It includes the following topics:

Overview of String Methods

The value property of a string object describes a sequence of text characters. In this topic, the term *string* represents the value of an instance of the string object. Other properties of the string object describe the string value and methods of the string object that manipulate the string value.

To indicate that a text literal is a string, you enclose it with quotation marks. In the following example, the first statement places the hello string in the word variable. The second statement sets the word variable to have the same value as the hello variable:

```
var word = "hello";
word = hello;
```

To declare a string you can use single quotes instead of double quotes. No difference exists between these quotes in Siebel eScript.

This topic uses the following formats:

- **stringVar.** Indicates a string variable. To use a property or to call a method, a specific instance of a variable must precede the period.
- **String name.** Indicates a static method of the string object. It does not apply to a specific instance of the string object.

How Siebel eScript Handles Special Characters In a String

A quotation mark is an example of a special character. To use a special character in a string, you must use a specific combination of characters that represent the special character. This combination allows Siebel CRM to understand how you intend it to use the character. For example, a quotation mark that is part of a string or a quotation mark that marks the end of the string.

Table 35 lists the character combinations that represent special characters. You cannot write code that uses these character combinations in a string that is enclosed by back quotes. For more information, see "Back Quote Usage in a String" on page 89.

Table 35. Character Combinations That Represent Special Characters

Character Combination	Special Character That the Character Combination Represents
\a	Audible bell.
\b	Backspace.
\f	Form feed.
\n	Newline.
\r	Carriage return.
\t	Tab.
\v	Vertical tab.
\'	Single quote.
\"	Double quote.
\\	Backslash character.
\0###	Octal number. For example: '\033' is the octal number.
\x##	Hex number. For example: '\x1B' is the hex number.
\0	Null character. For example: '\0' is the null character.
\u####	Unicode number. For example: '\u001B' is the Unicode number.

Back Quote Usage in a String

To configure Siebel eScript to not translate a character combination that typically represents a special character, you can use the following back quote:

```
`
```

If you use the back quote, then Siebel eScript interprets the character combination as a part of the string. For example, the following code lines illustrate different ways to reference a file name:

```
"c:\\autoexec.bat" // traditional C method
'c:\\autoexec.bat' // traditional C method
`c:\autoexec.bat'   // alternative Siebel eScript method
```

If a string includes a back quote, then you cannot include a special character that is represented by a back slash followed by a letter in that string. For example, \n.

Most versions of JavaScript do not support a string that includes a back quote. If you plan to use your script in some form of JavaScript other than Siebel eScript, then do not use back quotes.

Change String to Lowercase Method

The Change String to Lowercase method modifies every character that resides in the stringVar variable that is in uppercase to the lowercase equivalent. It returns a copy of this string that includes all lowercase characters.

Format

stringVar.toLowerCase()

Example

The following example assigns the value e. e. cummings to the variable poet:

```
var poet = "E. E. Cummings";
poet = poet.toLowerCase();
```

Change String to Uppercase Method

The Change String to Uppercase method modifies every character that resides in the stringVar variable that is in lowercase to the uppercase equivalent. It returns a copy of this string that includes all uppercase characters.

Format

stringVar.toUpperCase()

Example

The following example accepts a file name as input and displays it in uppercase:

```
   var filename = "c:\\temp\\trace.txt";;
   TheApplication().RaiseErrorText("The filename in uppercase is "
+filename.toUpperCase());
```

Create String From Substring Method

The Create String From Substring method returns a new string. Note the following:

- This string includes characters that the stringVar variable contains according to the start position and the end position that you specify.
- The length of this new string is equal to the value of the end argument minus the value of the start argument.
- It does not return the character that resides at the end position. If you do not specify the end argument, then it returns the characters from the value you specify in the start argument to the end of the string that resides in the stringVar variable.

Format

stringVar.substring(*start*[, *end*])

Table 36 describes the arguments for the Create String From Substring method.

Table 36. Arguments for the Create String From Substring Method

Argument	Description
start	An integer that identifies the starting position of the string that this method returns.
end	An integer that identifies the ending position plus 1 of the last character of the string that this method returns.

Example

For an example, see "Replace String Method" on page 97.

Create String From Unicode Values Method

The Create String From Unicode Values method converts Unicode values to a string. It uses Unicode values that you specify to determine the characters that are part of the string that it creates.

The String name is a property of the String constructor, so you use with this method instead of the variable name that you use with an instance method. Siebel eScript assumes that the values in the arguments that it passes to this method are Unicode values.

For more information, see "Clib Convert Character to ASCII Method" on page 288.

Format

`String.fromCharCode(`*`code1, code2, ... code`*`n)`

Table 37 describes the arguments for the Create String From Unicode Values method.

Table 37. Arguments for the Create String From Unicode Values Method

Argument	Description
code1, code2, ... coden	Each argument is an integer that identifies a Unicode code number.

Example 1

The following example sets the string1 variable to AB:

```
var string1 = String.fromCharCode(0x0041,0x0042);
```

Example 2

The following example uses the decimal Unicode values of the characters to create a string:

```
var seblStr = String.fromCharCode(83, 105, 101, 98, 101, 108);
```

This example provides a string that contains the following characters:

```
Siebel
```

For another example, see "Write Byte to Buffer Method" on page 118.

Get Character From String Method

The Get Character From String method returns the character that resides at a specific location in a string. The length of this character is 1.

To get the first character in a string, you use position 0. For example:

```
var string1 = "a string";
var firstchar = string1.charAt(0);
```

To get the last character in a string, you use length minus 1. For example:

```
var lastchar = string1.charAt(string1.length - 1);
```

If the value in the position argument is not between 0 and the value of stringVar.length minus 1, then this method returns an empty string.

Format

`stringVar``.charAt(`*`position`*`)`

Table 38 describes the arguments for the Get Character From String method.

Table 38. Arguments for the Get Character From String Method

Argument	Description
position	An integer that describes the position in the string of the character that this method returns. The position of the first character in the string is 0.

Get Unicode Character From String Method

The Get Unicode Character From String method returns the Unicode value of the character that resides at a specific position in a string. It returns a 16-bit integer between 0 and 65535. The value of the position argument identifies this position. If no character exists at this position, then it returns the following value:

NaN

This method uses the same arguments as the Get Regular Expression From String method. For more information, see Table 38 on page 92. For more information, see the following topics:

- "Clib Convert Character to ASCII Method" on page 288
- "Change String to Lowercase Method" on page 89
- "Create String From Unicode Values Method"

Format

stringVar.charCodeAt(*position*)

Usage

To get the first character in a string, you use position 0. For example:

```
var string1 = "a string";
string1.charCodeAt(0);
```

To get the last character in a string, you use length minus 1. For example:

```
string1.charCodeAt(string1.length - 1);
```

If the value in the position argument is not between 0 and the value of stringVar.length minus 1, then the Get Unicode Character From String method returns an empty string.

Example

The following eScript code configures Siebel CRM to allow the user to only enter characters that are part of the Latin character set. These characters must possess a Unicode value of less than 128. The user enters these characters in the First Name field. You add this code to the Contact business component. The Get Unicode Character From String method evaluates the Unicode value of each character that the user enters in the field that the FieldValue argument specifies:

```
function BusComp_PreSetFieldValue (FieldName, FieldValue)
{
// prevent non latin characters in First Name field
if (FieldName == "First Name")
     {
     for (var i=0;i<FieldValue.length;i++)
          {
          var co = FieldValue.charCodeAt(i);
          if (co > 127)
             {
             TheApplication().RaiseErrorText("Only characters from latin character
set are allowed!");
             }
          }
     }

return (ContinueOperation);
}
```

Get Regular Expression From StringVar Method

The Get Regular Expression From StringVar method searches stringVar for a regular expression. It returns one of the following:

- If it finds a match, then it returns an array of strings that includes information about each string and the property sets for these strings.
- If it does not find the regular expression, then it returns the following value:

  Null

Format

stringVar.match(*regexp*)

Table 39 describes the arguments for the Get Regular Expression From StringVar method.

Table 39. Arguments for the Get Regular Expression From StringVar Method

Argument	Description
regexp	A regular expression that you describe as a literal or as a variable.

Usage without Setting the Global Attribute

If you use the Get Regular Expression From StringVar method with the g global attribute not set on the regular expression, then this usage is the same as with the Get Regular Expression From String method. For more information, see "Usage Without Setting the Global Attribute" on page 198 and "Get Regular Expression from String Method" on page 197.

Usage with Setting the Global Attribute

If you use the Get Regular Expression From StringVar method, and if you set the g global attribute on the regular expression, and if this method finds a match, then it does the following:

- Returns element 0 of the return array as the first text in the string that matches the primary pattern of the regular expression.
- Returns each subsequent element of the return array as the next text in the string that matches the primary pattern of the regular expression, and that starts after the last character of the previous match. It does not return any matches that overlap other matches.

For example, assume the following is true:

- The primary pattern of the regular expression is a.. (the letter a followed by any two characters).
- The string is abacadda.

In this situation the return array includes the following:

- aba
- add

It does not include aca.

If you set the g global attribute on the regular expression, then usage for the Get Regular Expression From StringVar method is very different than usage for the Get Regular Expression From String method.

For more information, see "Get Regular Expression from String Method" on page 197.

Example 1

The following example uses the Get Regular Expression From StringVar method with a regular expression whose global attribute is not set:

```
function fn ()
{
   var myString = new String("Better internet");
   var myRE = new RegExp(/(.).(.er)/i);
   var results = myString.match(myRE);
   var resultmsg = "";
   for(var i =0; i < results.length; i++)
   {
      resultmsg = resultmsg + "return[" + i + "] = " + results[i] + "\n";
   }
   TheApplication().RaiseErrorText(resultmsg);
}
fn ();
```

This example provides the following output:

```
return[0] = etter  \\First text that contains primary pattern ...er (any three
                   \\characters followed by "er")
return[1] = e       \\First text that matches the first subpattern (.) (any single
                   \\character) in the first text that matches the primary pattern
```

```
return[2] = ter     \\First text that matches the second subpattern (.er) (any single
                    \\character followed by "er") in the first text that matches
                    \\the primary pattern
```

Example 2

The following example uses the Get Regular Expression From StringVar method with a regular expression whose global attribute is set. The method returns matches of the primary pattern of the regular expression that do not overlap:

```
function fn ()
{
   var str = "ttttot tto";
   var pat = new RegExp("t.t", "g");
   var rtn = str.match(pat);
   var resultmsg = "";
   for(var i =0; i < rtn.length; i++)
   {
      resultmsg = resultmsg + "match [" + i + "] = " + rtn[i] + "\n";
   TheApplication().RaiseErrorText(resultmsg);
   }
}
fn ();
```

This code produces the following output. This output does not include the ttt instance that starts at position 1 or the t t instance because these instances start in other strings that the Get Regular Expression From StringVar method returns:

```
match [0] = ttt
match [1] = tot
```

Get String Length Method

The Get String Length method returns an integer that describes the length of the string.

Format

stringVar.length

Example 1

The following example displays the number 14, which is the number of characters in the string. The position of the last character in the string is equivalent to the value in stringVar.length minus 1, because the position begins at 0, not at 1:

```
var string1 = "No, thank you.";
TheApplication().RaiseErrorText(string1.length);
```

Example 2

The following example returns the length of a name that the user enters, including spaces:

```
var userName = "Christopher J. Smith";
TheApplication().RaiseErrorText( "Your name has " +
   userName.length + " characters.");
```

Parse String Method

The Parse String method parses a string into an array of strings according to the delimiters that you specify in the delimiter argument. Note the following:

- It returns an array of strings, each of which begins at an instance of the delimiter character.
- It does not include the delimiter in any of the strings.
- If you do not specify the delimiter argument or if this argument contains an empty string (""), then it returns an array of one element, which includes the original string.
- It is the inverse of *arrayVar*.join.

For more information, see "Create Array Elements Method" on page 80.

Format

stringVar.split([*delimiter*])

Table 40 describes the arguments for the Parse String method.

Table 40. Arguments for the Parse String Method

Argument	Description
delimiter	The character where this method splits the value stored in stringVar.

Example

The following example splits a typical Siebel command line into separate elements. It creates a separate array element at each space character. You must configure Siebel CRM to modify the string with character combinations so that Siebel eScript can understand it. The cmdLine variable must occur on a single line. In this book this variable wraps to a second line:

```
function Button3_Click ()
{
   var msgText = "The following items occur in the array:\n\n";
   var cmdLine = "C:\\Siebel\\bin\\siebel.exe /c
\'c:\\siebel\\bin\\siebel.cfg\' /u SADMIN /p SADMIN /d Sample"
   var cmdArray = cmdLine.split(" ");
   for (var i = 0; i < cmdArray.length; i++)
      msgText = msgText + cmdArray[i] + "\n";
   TheApplication().RaiseErrorText(msgText);
}
```

This example produces the following result:

```
The following items occur in the array:
C:\Siebel\bin\siebel.exe
/c
'C:\siebel\bin\siebel.cfg'
/u
SADMIN
/p
SADMIN
/d
Sample
```

Replace String Method

The Replace String method uses the regular expression that you define in the pattern argument to search a string. If it finds a match, then it replaces the string it finds with the string that you define in the replexp argument.

If you use T eScript code, and if this method replaces a string, then it sets the appropriate static properties for the regular expression object. These properties provide more information about the replacements. For more information, see "Using the Get Regular Expression from String Method with the T eScript Engine" on page 199.

Format

stringVar.replace(*pattern*, *replexp*)

Table 41 describes the arguments for the Replace String method.

Table 41. Arguments for the Replace String Method

Argument	Description
pattern	Regular expression that this method finds in a string.
replexp	A replacement expression that can include one of the following items: ■ String ■ String that includes regular expression elements ■ Function

Special Characters You Can Use in a Replacement Expression

Table 42 describes the special characters that you can use in a replacement expression.

Table 42. Special Characters You Can Use in a Replacement Expression

Character	Description
$1, $2 … $9	The text that the regular expression matches. This text resides in a set of parentheses in the string. For example, $1 replaces the text that the Replace String method matches in the first regular expression it encounters that resides in a set of parentheses.
$+	Same as the $1, $2 … $9 characters, except with the $+ character the replacement occurs in the regular expression that resides in the last set of parentheses.
$&	The text that a regular expression matches.
$`	The text to the left of the text that a regular expression matches.
$'	The text to the right of the text that a regular expression matches.
\$	The dollar sign character.

Example

The following example includes the Replace String method:

```
var rtn;
var str = "one two three two one";
var pat = /(two)/g;

// rtn == "one zzz three zzz one"
rtn = str.replace(pat, "zzz");

// rtn == "one twozzz three twozzz one";
rtn = str.replace(pat, "$1zzz");

// rtn == "one 5 three 5 one"
rtn = str.replace(pat, five());

// rtn == "one twotwo three twotwo one";
rtn = str.replace(pat, "$&$&”);

function five() {
   return 5;

}
```

Search String for Substring Method

The Search String for Substring method searches the stringVar variable for the entire string that you specify in the substring argument. It returns the position of the first occurrence of this string.

If any of the following situations is true, then it returns a value of negative 1:

- It does not find the value that you specify in the substring argument.
- The value you specify in the offset argument is outside the range of positions in the string.

Values you enter for arguments are case-sensitive.

Format

stringVar.indexOf(*substring* [, *offset*])

Table 43 describes the arguments for the Search String for Substring method.

Table 43. Arguments for the Search String for Substring Method

Argument	Description
substring	One or more characters that this method searchs.The substring argument can contain a single character.
offset	The position in the string where this method starts searching. This method does the following according to how you set the offset argument: ■ **You specify the offset argument.** Starts searching at the position that you specify in the offset argument. ■ **You do not specify the offset argument.** Starts searching at position 0.

Example 1

The following example returns the position of the first a character that occurs in the string. In this example, the first a character occurs at position 2:

```
var string = "what a string";
var firsta = string.indexOf("a")
```

Example 2

The following example returns 3, which is the position of the first a character in the string when starting from the second character of the string:

```
var magicWord = "abracadabra";
var secondA = magicWord.indexOf("a", 1);
```

Related Topics

For more information, see the following topics:

- “Clib Search String for Character Method” on page 258
- “Clib Search String for Character Set Method” on page 259
- “Search String for Last Substring Method” on page 100

Search String for Last Substring Method

The Search String for Last Substring method searches the stringVar variable for the string that you specify in the substring argument. It returns the position of the last occurrence of this string in a string.

If any of the following situations is true, then it returns a value of negative 1:

- It does not find the value that you specify in the substring argument.
- The value you specify in the offset argument is outside the range of positions in the string.

Note the following:

- To limit the search to a string of leftmost characters of the string, you can use the offset argument.
- The first character of the substring must occur at a position that is no greater than the offset.
- If the value you specify in the substring argument does not occur entirely before or at the offset, then the Search String for Last Substring method still returns the position of the substring that it finds.

This method uses the same arguments as the Search String for Substring method. For more information, see Table 43 on page 99.

Format

stringVar.lastIndexOf(*substring* [, *offset*])

This method does the following according to how you specify the offset argument:

- **You do specify the offset argument.** It searches the string up to the position that you specify in the offset argument, and then returns the rightmost position where the substring begins. It does not consider any substring that occurs after the position that you specify in the offset argument.
- **You do not specify the offset argument.** It returns the rightmost position in the entire string where the substring begins.

Example 1

The following example returns the position of the last occurrence of the a character in the string. It returns a value of 5:

```
var string = "what a string";
string.lastIndexOf("a")
```

Example 2

The following example returns the position of the last occurrence of the abr string, beginning at a position that is not greater than 8. It returns a value of 7:

```
var magicWord = "abracadabra";
var lastabr = magicWord.lastIndexOf("abr", 8);
```

Search StringVar for Regular Expression Method

The Search StringVar for Regular Expression method searches a string for a regular expression. It returns one of the following:

- If it finds the regular expression, then it returns the position of this regular expression.
- If it does not find the regular expression, then it returns negative 1.

You can write code that runs this method in server script or browser script.

This method uses the same argument as the Get Regular Expression From StringVar method. For more information, see Table 39 on page 93.

Format

stringVar.search(*regexp*)

Example

The following example uses the Search StringVar for Regular Expression method:

```
function Test(sValue)
{
   //Validate for 5 digit numbers
     var sCheck = /^\d{5}$/;  //regular expression defining a 5 digit number
     if(sValue.search(sCheck)==0)
  {
        return("Valid");
     }
     else
     {
        return("Invalid");
  }
}
```

BLOB Methods

This topic includes the following topics:

About the BLOB Descriptor

The blobDescriptor object describes the structure of a BLOB (binary large object). If you must configure Siebel CRM to send an object to a process other than the Siebel eScript interpreter, such as to a Windows API function, then you must configure it to create a blobDescriptor object that describes the order and type of data of this object. This description describes how to store the properties of the object in memory. You use it with methods such as the Siebel Library Call DLL method or the Clib Read From File method. For more information, see "Siebel Library Call DLL Method" on page 201 and "Clib Read From File Method" on page 240.

A BLOB descriptor includes the same data properties as the object it describes. You must set a value for each property that specifies how much memory is required to store the data that the property holds. To refer to the arguments passed to the constructor function, you use the following keyword:

```
this
```

You can think of this keyword conceptually as *this object*. Consider the following object:

```
Rectangle(width, height)
{
   this.width = width;
   this.height = height;
}
```

To configure Siebel eScript to pass data to the following items, you typically use a BLOB descriptor:

- Siebel eScript data structure, which is similar to JavaScript
- C program or a C++ program
- Clib method

These items expect a rigid and precise description of the values that Siebel eScript passes.

Example of Using a BLOB Descriptor

The following example creates a blobDescriptor object that describes the Rectangle object:

```
var bd = new blobDescriptor();

bd.width  = UWORD32;
bd.height = UWORD32;
```

In this example, you can use Siebel eScript to pass the bd variable as a blobDescriptor argument to a function that requires a blob descriptor. The values set for the properties depend on what the receiving function expects. In this example the function that Siebel CRM calls expects to receive an object that includes two 32-bit words or data values. If you write a BLOB descriptor for a function that expects to receive an object that contains two 16-bit words, then set the value for the two properties to UWORD16.

Values You Must Use with a BLOB Descriptor

Table 44 describes the values that you must use with blobDescriptor object properties. To indicate the number of bytes that are required to store the property, you use one of these values. If the BLOB descriptor describes an object property that is a string, then you must set the corresponding property to a numeric value that is larger than the length of the longest string that the property can hold. You can write code that omits an object method from a BLOB descriptor.

Table 44. Values You Can use With the blobDescriptor

Value	Description
WCHAR	Handled as a native Unicode string.
UWORD8	Stored as an unsigned byte.
SWORD8	Stored as an integer.
UWORD16	Stored as an unsigned 16-bit integer.
SWORD16	Stored as a signed 16-bit integer.
UWORD24	Stored as an unsigned 24-bit integer.
SWORD24	Stored as a signed 24-bit integer.
UWORD32	Stored as an unsigned 32-bit integer.
SWORD32	Stored as a signed 32-bit integer.
FLOAT32	Stored as a floating-point number.
FLOAT64	Stored as a double-precision floating-point number.
STRINGHOLDER	Indicates a value that Siebel eScript saves in a string. Siebel eScript passes this value to a function. This function saves this string. Siebel eScript does the following work: 1 Allocates 10,000 bytes to contain the string. 2 Truncates this length to the appropriate size. 3 Removes any terminating null characters. 4 Initializes the properties of the string.

Get BLOB Data Method

This Get BLOB Data method reads data from a binary large object. It returns the data from the BLOB.

Format A

```
Blob.get(blobVar, offset, dataType)
```

You use format A for byte, integer, or float data.

Format B

Blob.get(*blobVar, offset, bufferLen*)

You use format B for byte data.

Format C

Blob.get(*blobVar, offset, blobDescriptor dataDefinition*)

You use format C for object data.

Arguments

Table 45 describes the arguments for the Get BLOB Data method.

Table 45. Arguments for the Get BLOB Data Method

Argument	Description
blobVar	The name of the binary large object that this method manipulates.
offset	The position in the BLOB that Siebel CRM uses to read the data.
dataType	An integer value that identifies the data format in the BLOB. The dataType argument must include one of the values you must use with a BLOB descriptor. For more information, see "Values You Must Use with a BLOB Descriptor" on page 103.
bufferLen	An integer that specifies the size of the buffer in bytes.
blobDescriptor dataDefinition	A blobDescriptor object that identifies the data format in the BLOB.

Example

The following example describes how to get values from a BLOB object:

```
function GetBlobVal()
{
   var a, b, c;
   a = "";
   b = 1234;
   c = 12345678;
   // Call a function to build the Blob
   var blob = BuildBlob(a, b, c);
   TheApplication().TraceOn("c:\\temp\\blob.txt","Allocation","All");
   // Get the values from the blob object
   // The first variable is string
   var resultA = Blob.get(blob,0,1000);
   // The second variable is an integer
   var resultB = Blob.get(blob,1000,UWORD16);
   // The third variable has a type of float
   var resultC = Blob.get(blob,1002,FLOAT64);
```

```
   TheApplication().Trace(resultA);
   TheApplication().Trace(resultB);
   TheApplication().Trace(resultC);
}

function BuildBlob(a, b, c)
{

   var blob;
   a = "Blob Test Value From Function";
   var offset = Blob.put(blob, 0, a, 1000);
   offset = Blob.put(blob, offset, b*2, UWORD16);
   Blob.put(blob, offset, c*2, FLOAT64);
   return blob;
}
```

Get BLOB Size Method

The Get BLOB Size method determines the size of a BLOB object. It returns the number of bytes that this BLOB object contains. It returns this value in the blobVar argument.

Format A

`Blob.size(`*`blobVar`*`[, `*`SetSize`*`])`

Format B

`Blob.size(`*`dataType`*`)`

Format C

`Blob.size(`*`bufferLen`*`)`

Format D

`Blob.size(`*`blobDescriptor dataDefinition`*`)`

Arguments

Table 46 describes the arguments for the Get BLOB Size method. If you specify any of the following arguments, then Siebel CRM uses the values you specify to convert Siebel eScript data to a BLOB, and to covert BLOB data to Siebel eScript data:

- dataType
- bufferLen

- dataDefinition

Table 46. Arguments for the Get BLOB Size Method

Argument	Description
blobVar	The name of the binary large object that this method examines.
setSize	An integer that determines the size of the BLOB. If you specify the SetSize argument, then this method does the following work: ■ Modifies the size of the BLOB that you identify in the blobVar argument to the value you specify in the SetSize argument ■ Returns a value in the setSize argument
dataType	An integer value that describes the format of the data in the BLOB. The dataType argument must include one of the values that you use with a BLOB descriptor. For more information, see "Values You Must Use with a BLOB Descriptor" on page 103.
bufferLen	An integer that describes the number of bytes in the buffer.
blobDescriptor dataDefinition	A blobDescriptor object that describes the format of the data in the BLOB.

Write BLOB Data Method

The Write BLOB Data method writes data to a binary large object. It returns an integer that identifies the byte offset of the byte that occurs after the end of the data that this method writes. If it writes data at the end of the BLOB, then this integer identifies the size of the BLOB.

You can write code that adds data at any position in a BLOB. The data length is variable. This method does not pad each data element with null values as a way to make every data element a uniform length. The exact length depends on the CPU. Thirty two bytes is a common length.

For more information, see "Usage of the Term Put" on page 76.

Format A

```
Blob.put(blobVar[, offset], data, dataType)
```

Format B

```
Blob.put(blobVar[, offset], buffer, bufferLen)
```

Format C

```
Blob.put(blobVar[, offset], srcStruct, blobDescriptor dataDefinition)
```

To pass the contents of an existing BLOB that resides in the srcStruct argument to the blobVar argument, you can use format C.

Arguments

Table 47 describes the arguments for the Write BLOB Data method.

Table 47. Arguments for the Write BLOB Data Method

Argument	Description
blobVar	The name of the binary large object that this method manipulates.
offset	The position in the BLOB where this method adds data. If you do not provide a value for the offset argument, then this method does one of the following depending on if the BLOB is defined: ■ **BLOB is defined.** Adds data at the end of the BLOB. ■ **BLOB is not defined.** Adds data at offset 0.
data	The data that this method writes.
dataType	The format of the data in the BLOB. This method converts the data to the format that you specify in the dataType argument, and then copies this data to the position that you specify in the offset argument. If the value that you specify in the dataType argument is not the length of a byte buffer, then the dataType argument must include one of the values you use with a BLOB descriptor. For more information, see "Values You Must Use with a BLOB Descriptor" on page 103.
buffer	A variable that contains a buffer.
bufferLen	An integer that specifies the buffer length.
srcStruct	A BLOB that contains the data that this method writes.
blobDescriptor dataDefinition	A blobDescriptor object that describes the format of the data in the BLOB.

Example

Assume you send a data pointer to an external C library. Assume the library expects data in the following packed C structure:

```
struct foo
{
   signed char a;
   unsigned int b;
   double c;
};
```

The following example creates a structure from three corresponding variables and returns the offset of the next available byte:

```
function BuildFooBlob(a, b, c)
{
   var offset = Blob.put(foo, 0, a, SWORD8);
   offset = Blob.put(foo, offset, b, UWORD16);
   Blob.put(foo, offset, c, FLOAT64);
   return foo;
}
```

The following example creates a structure from three corresponding variables but does not include an offset:

```
functionBuildFooBlob(a, b, c)
{
   Blob.put(foo, a, SWORD8);
   Blob.put(foo, b, UWORD16);
   Blob.put(foo, c, FLOAT64);
   return foo;
}
```

Buffer Methods

This topic describes buffer methods. It includes the following topics:

Overview of Buffer Methods

A buffer method allows you to manipulate data at a very basic level. It is required if the relative position of data in memory is important. You can configure Siebel CRM to store any type of data in a buffer object.

You can configure Siebel CRM to create a new buffer object from the following items:

- **Nothing.**
- **A string, a buffer, or a buffer object.** Siebel CRM copies the contents of the string, buffer, or buffer object to the new buffer object.

The examples for buffer methods in this chapter use the bufferVar argument as a generic argument name. Siebel CRM assigns a buffer object to this generic argument.

About Buffer Constructors

This topic describes the formats you can use to create a buffer object.

Table 48 describes the buffer constructor arguments that are common to formats A, B, C, and D.

Table 48. Arguments for the Buffer Constructor that Are Common to Formats A, B, C, and D

Argument	Description
unicode	You can use one of the following values: ■ **True.** Siebel eScript creates the new buffer as a Unicode string regardless of whether the input string is Unicode or not. ■ **False.** Siebel eScript creates the new buffer as an ASCII string regardless of whether the input string is Unicode or not. False is the default value.
bigEndian	You can use one of the following values: ■ **True.** Siebel eScript stores the largest data values in the most significant byte. ■ **False.** Siebel eScript stores the largest data values in the least significant byte. False is the default value.

Format A

```
new Buffer([size] [, unicode] [, bigEndian]);
```

Table 49 describes the buffer constructor arguments that are specific to format A.

Table 49. Arguments for the Buffer Constructor for Format A

Argument	Description
size	The size of the new buffer that Siebel eScript creates. You can do one of the following: ■ **Specify the size argument.** Siebel eScript creates the new buffer with the size you specify and fills it with null bytes. ■ **Do not specify the size argument.** Siebel eScript creates the new buffer with a size of 0. You can configure it to dynamically extend the new buffer later.

Format B

```
new Buffer( string [, unicode] [, bigEndian] );
```

Format B creates a new buffer object from a string that you provide. A line of code that uses format B creates a new buffer object from the buffer provided. Siebel CRM copies the contents of the buffer into the new buffer object. The unicode argument and the bigEndian argument do not affect this conversion, although they do set the relevant flags for future use.

Table 50 describes the buffer constructor arguments that are specific to format B.

Table 50. Arguments for the Buffer Constructor for Format B

Argument	Description
string	The string that Siebel eScript uses as input to create the buffer. If the string argument contains a Unicode string, then Siebel eScript creates the buffer as a Unicode string. To use a Unicode string, you must enable Unicode in the Siebel application. To override this behavior, you can specify false in the optional unicode argument. The size of the buffer depends on if the string is a Unicode string: ■ **The string is a Unicode string.** The size of the buffer is twice the length of the input string. ■ **The string is not a Unicode string.** The size of the buffer is the length of the input string. A buffer constructor does not add a terminating null byte at the end of the string.

Format C

```
new Buffer(buffer [, unicode] [, bigEndian]);
```

Table 51 describes the buffer constructor arguments that are specific to format C.

Table 51. Arguments for the Buffer Constructor for Format C

Argument	Description
buffer	The buffer that Siebel eScript uses as input to create the new buffer.

Format D

new Buffer(*bufferobject*);

A line of code that uses format D creates a new buffer object from another buffer object. Siebel CRM copies the contents of the buffer object to the new buffer verbatim, including the cursor position, size, and data.

Table 52 describes the buffer constructor arguments that are specific to format D.

Table 52. Arguments for the Buffer Constructor for Format D

Argument	Description
bufferobject	The buffer object that Siebel eScript uses as input to create the new buffer.

Example

The following example creates new buffer objects:

```
function BufferConstruct()
{
   TheApplication().TraceOn("c:\\temp\\BufferTrace.doc","Allocation","All");
   // Create empty buffer with size 100
   var buff1 = new Buffer(100 , true , true);
   // Create a buffer from string
   var buff2 = new Buffer("This is a buffer String constructor example", true);
   // Create buffer from buffer
   var buff3 = new Buffer(buff2,false);
   try
   {
      with(buff1)
      {
         // Add values from 0-99 to the buffer
         for(var i=0;i<size;i++)
         {
            putValue(i);
         }
         var val = "";
         cursor=0;
         // Read the buffer values into variable
         for(var i=0;i<size;i++)
         {
            val += getValue(1)+" ";
         }
```

```
            // Trace the buffer value
            TheApplication().Trace("Buffer 1 value: "+val);
         }
         with(buff2)
         {
            // Trace buffer 2
            TheApplication().Trace("Buffer 2 value: "+getString());
         }
         // Trace buffer 3
         with(buff3)
         {
            TheApplication().Trace("Buffer 3 value: "+getString());
         }
      }
      catch(e)
      {
         TheApplication().Trace(e.toString());
      }
   }
```

Create Buffer Method

The Create Buffer method extracts the data that exists between two positions in a buffer. It returns this data in a new buffer object. This method does the following:

- If the value that the beginning argument contains is less than 0, then it treats this value as 0, which is the beginning of the buffer.
- If the value that the end argument contains is beyond the end of the buffer, then it uses null bytes to increase the size of the new buffer. It does not modify the source buffer. It duplicates the values of the unicode argument and the bigEndian argument in the new buffer.
- Sets the length of the new buffer to the value that the end argument contains minus the value that the beginning argument contains.

Format

bufferVar.subBuffer(*beginning*, *end*)

Table 53 describes the arguments for the Create Buffer method.

Table 53. Arguments for the Create Buffer Method

Argument	Description
beginning	The position in the source buffer where this method begins to extract data.
end	The position in the source buffer where this method stops extracting data.

How the Create Buffer Method Sets the Cursor

Table 54 describes how the Create Buffer method sets the cursor.

Table 54. How the Create Buffer Method Sets the Cursor

Original Cursor Position	How the Create Buffer Method Sets the Cursor
Between the value that the beginning argument contains and the value that the end argument contains.	Sets the cursor position to a new relative position in the new buffer.
Before the value that the beginning argument contains.	Sets the cursor position to 0 in the new buffer.
After the value that the end argument contains.	Sets the cursor position to the end of the new buffer.

Example

The following example creates a new buffer named language and displays the contents of this buffer in a string named Siebel eScript. The Siebel eScript text begins in the nineteenth position:

```
var loveIt= new Buffer("I love coding with Siebel eScript!");
var language = loveIt.subBuffer(19, (loveIt.size - 1))
TheApplication().RaiseErrorText(language);
```

Related Topics

For more information, see "Get String From Buffer Method" on page 115.

Get Buffer Data Method

The Get Buffer Data method returns a string that contains the same data that the buffer contains. If necessary, it does a Unicode conversion according to the value of the Use Unicode in Buffer property. For more information, see "Avoiding an Exception Error That Is Not Handled" on page 70.

Format

bufferVar.toString()

Example

The following example uses the Get Buffer Data method:

```
try
{
   do_something;
}
catch( e )
{
```

```
      TheApplication().RaiseErrorText(Clib.rsprintf(
         "Something bad happened: %s\n",e.toString()));
   }
```

Get Cursor Position Value From Buffer Method

The Get Cursor Position Value From Buffer method returns the value that a position contains from a buffer. This position is the position where the cursor currently resides. To determine where to read from the buffer, you can use the Cursor property. For more information, see "Cursor Position in Buffer Property" on page 119.

Format

bufferVar.getValue([*valueSize*][, *valueType*])

Table 55 describes the arguments for the Get Cursor Position Value From Buffer method.

Table 55. Arguments for the Get Cursor Position Value From Buffer Method

Argument	Description
valueSize	A positive number that describes the number of bytes that the this method reads. You can use any of the following values: ■ 1 ■ 2 ■ 3 ■ 4 ■ 8 ■ 10 The default value is 1. These values must not conflict with any values you use with the valueType argument. For more information, see "Values You Can Use with the ValueSize and ValueType Arguments" on page 115.
valueType	The type of data that the this method reads. You can use one of the following values: ■ Signed ■ Unsigned ■ Float Signed is the default value.

Values You Can Use with the ValueSize and ValueType Arguments

Table 56 describes the value combinations you can use with the valueSize argument and the valueType argument. The values of the valueSize argument and the valueType argument must match the structure of the data that the Get Cursor Position Value From Buffer method reads. Any other combination causes an error.

Table 56. Value Combinations You Can Use with Arguments of the Get Cursor Position Value From Buffer Method

Value in the valueSize Argument	Value in the valueType Argument
1	signed, unsigned
2	signed, unsigned
3	signed, unsigned
4	signed, unsigned, float
8	float

Get String From Buffer Method

The Get String From Buffer method returns a string that starts at the current cursor position in a buffer and continues for the number of bytes that you specify in the length argument. It reads the string according to the value of the unicode flag of the buffer. It does not add a terminating null byte even if you do not provide a length argument.

Format

bufferVar.getString([*length*])

Table 57 describes the arguments for the Get String From Buffer method.

Table 57. Arguments for the Get String From Buffer Method

Argument	Description
length	The length of the string to return, in bytes. If you do not specify the length argument, then this method reads the data until it encounters one of the following items: ■ A null byte ■ The end of the buffer

Put String in Buffer Method

The Put String in Buffer method replaces existing data in a buffer with a string that you specify. It replaces data starting at the current position of the cursor. This method does one of the following depending on if the Unicode flag in the buffer object is set:

- **Set.** It puts the string in the buffer object as a Unicode string. It increments the cursor by twice the length of the string.
- **Not set.** It puts the string in the buffer object as an ASCII string. It increments the cursor by the length of the string.

This method does not add a terminating null byte at end of the string.

To put a null string in the buffer object, you can use the following code:

```
buf1.putString("Hello");   // Put the string into the buffer
buf1.putValue( 0 );         // Add terminating null byte
```

Format

bufferVar.putString(*string*)

Table 58 describes the arguments for the Put String in Buffer method.

Table 58. Arguments for the Put String in Buffer Method

Argument	Description
string	The string literal that this method puts in the buffer object, or the string variable whose value it puts in the buffer object.

Example

The following example places the language string in the exclamation buffer and displays the modified contents of the explanation buffer:

```
function eScript_Click ()
{
   var exclamation = new Buffer("I enjoy coding with . . .");
   var language = "Siebel eScript.";
   exclamation.cursor = 20;
   exclamation.putString(language);
   TheApplication().RaiseErrorText(exclamation);
}
```

This modification is a string that contains the following value:

```
I enjoy coding with Siebel eScript.
```

Related Topics

For more information, see the following topics:

- "Usage of the Term Put" on page 76.

■ “Get String From Buffer Method” on page 115.

Put Value in Buffer Method

The Put Value in Buffer method replaces existing data in a buffer with a value that you specify. It replaces data starting at the current position of the cursor. It puts the value that the buffer contains at the current cursor position, and then automatically increments the cursor value by the value that the value argument contains.

To put a value at a specific position and preserve the cursor position, you can add code that is similar to the following:

```
var oldCursor = bufferItem.cursor; // Save the cursor position
bufferItem.cursor = 20;            // Set to new position
bufferItem.putValue(foo);            // Put bufferItem at offset 20
bufferItem.cursor = oldCursor        // Restore cursor position
```

For more information, see “Usage of the Term Put” on page 76.

Format

bufferVar.putValue(*value*[, *valueSize*][, *valueType*])

Table 59 describes the arguments for the Put Value in Buffer method.

Table 59. Arguments for the Put Value in Buffer Method

Argument	Description
value	A number.
valueSize	The description for the valueSize argument and the valueType argument for the Put Value in Buffer method is the same as the description for these arguments for the Get Cursor Position Value From Buffer method. For more information, see “Get Cursor Position Value From Buffer Method” on page 114.
valueType	

Avoiding Digit Loss

To put the value in the buffer, the Put Value in Buffer method uses byte ordering according to the current value that the bigEndian argument contains. If it puts a smaller float value, such as 4, then digits are lost. It converts a value such as 1.4 to a value that is approximately 1.39999974. This conversion is insignificant and you can ignore it.

Note the following example:

```
bufferItem.putValue(1.4,8,"float");
bufferItem.cursor -= 4;
if( bufferItem.getValue(4,"float") != 1.4 )
// This is not necessarily true due to significant digit loss.
```

To prevent this situation, you can set the valueSize argument to 8 instead of 4. You can use a valueSize of 4 for a floating-point value, but be aware that some digit loss might occur. This loss might not be significant enough to affect most calculations.

Write Byte to Buffer Method

The Write Byte to Buffer method writes a byte to a buffer at a position that you specify.

Format

bufferVar[*offset*]

Table 60 describes the arguments for the Write Byte to Buffer method.

Table 60. Arguments for the Write Byte to Buffer Method

Argument	Description
offset	A number that describes a position in the buffer that the bufferVar method identifies. This method does one of the following: ■ Places a byte at the offset position. ■ Reads data from the offset position. Note the following: ■ If the value that the offset argument contains is less than 0, then this method uses 0. ■ If the value that the offset argument contains is greater than the length of the buffer, then this method uses null bytes to increase the size of the buffer.

Usage

The Write Byte to Buffer method is an array-like version of the Get Cursor Position Value From Buffer method and the Put Value in Buffer method except that the Write Byte to Buffer method works only with bytes. You can write code that gets or sets these values. For example, the following code sets the goo variable to the value of a byte. This byte resides in the buffer at offset position 5:

```
goo = foo[5]
```

The following code sets the value of position 5 in the foo buffer to the value that the goo variable contains:

```
foo[5] = goo
```

This code assumes the value that the goo variable contains is a single byte value.

Every get or put operation uses eight-bit signed words (SWORD8). If you must work with character values, then you must convert these values to their ANSI equivalent or Unicode equivalent.

Related Topics

For more information, see the following topics:

- "Usage of the Term Put" on page 76.
- "Get Cursor Position Value From Buffer Method" on page 114
- "Put Value in Buffer Method" on page 117

Buffer Size Property

The Buffer Size property is the size of the buffer object. You can write code that sets a value for the Buffer Size property. For example:

```
inBuffer.size = 5
```

Siebel CRM does the following:

- If the buffer size increases beyond the current maximum size of the buffer, then it uses null bytes to fill the additional positions.
- If the buffer size decreases so that the cursor position is beyond the end of the buffer, then it moves the cursor position to the end of the modified buffer.

For more information, see "Cursor Position in Buffer Property" on page 119.

Format

bufferVar.size

Cursor Position in Buffer Property

The Cursor Position in Buffer property stores the current position of the buffer cursor. Note the following:

- The value of the cursor position is always between 0 and the value that the Buffer Size property contains.
- If you use Siebel eScript to set the cursor beyond the end of a buffer, then it increases the buffer to accommodate the new position. It fills the new positions with null bytes.
- If you use Siebel eScript to set the cursor to a value that is less than 0, then it places the cursor at position 0 of the buffer.

Format

bufferVar.cursor

Example

For examples, see "Get String From Buffer Method" on page 115 and "Create Buffer Method" on page 112.

Data in Buffer Property

The Data in Buffer property is a reference to the internal data of a buffer. You can write code that uses it as a temporary value to pass buffer data to a function that does not recognize a buffer object.

Format

bufferVar.data

Use Big Endian in Buffer Property

The Use Big Endian in Buffer property is a Boolean flag that specifies to use big endian byte ordering if Siebel CRM calls the Get Cursor Position Value From Buffer method or the Put Value in Buffer method. Siebel CRM stores bytes according to the following settings:

- **Use Big Endian in Buffer property is true.** Stores the bytes in descending order of significance.
- **Use Big Endian in Buffer property is false.** Stores the bytes in ascending order of significance.

Siebel CRM sets this value when it creates a buffer. You can configure it to modify this value at any time.

The Use Big Endian in Buffer property defaults to the state of the operating system and processor.

If a data value includes more than one byte, then the following occurs:

- The byte that contains the smallest units of the value is the least significant byte.
- The byte that contains the largest units of the value is the most significant byte.

Format

bufferVar.bigEndian

Use Unicode in Buffer Property

The Use Unicode in Buffer property is a Boolean flag that specifies whether to use a Unicode string when calling the Get String From Buffer method or the Put String in Buffer method. Siebel CRM sets the value for the Use Unicode property when it creates a buffer. You can configure it to modify this value. This property defaults to false.

Format

bufferVar.unicode

Example

The following example sets the Use Unicode in Buffer property of a new buffer to true:

```
var aBuffer = new Buffer();
aBuffer.unicode = true;
```

Date and Time Methods

This topic describes date and time methods. It includes the following topics:

Overview of Date Methods

Siebel eScript provides the following ways to work with dates:

- The standard date object in JavaScript.
- The Clib object that implements routines from the C programming language. For more information, see Chapter 6, "C Language Library Reference."

The following methods convert dates in the format of one date system to the format of the other date system:

- Date.fromSystem
- Date.toSystem

This chapter describes the JavaScript Date object.

To indicate the name of a variable that you create to hold a date value, this chapter uses dateVar.

Format for Calling a Date Method

To call a date method, you must precede the method name with a specific instance of a variable followed by a period. For example, assume you create a date object named aDate. To call the getDate method, you use the following format:

```
aDate.getDate
```

Siebel CRM uses a literal value to call a static method, such as Date.parse. The beginning of a static method includes the following format:

```
Date.
```

These methods are part of the date object instead of an instance of the Date object.

Caution About Using Two Digit Dates

Siebel eScript uses the ECMAScript standard for two digit dates, which might be different from the formats that other applications use, including Siebel CRM.

CAUTION: To prevent a year 2000 (Y2K) problem, avoid using a two digit date in your Siebel eScript code.

Values for Dates and Times

Table 61 describes values for months, days, hours, minutes, and seconds. Many Siebel eScript objects use these same values.

Table 61. Values for Months, Days, Hours, Minutes, and Seconds

Time Period	Description
month	A month, specified as an integer from 0 to 11. January is 0 and December is 11.
day	A day of the month, specified as an integer from 1 to 31. The first day of the month is 1. The last day of the month is 28, 29, 30, or 31.
hours	An hour, specified as an integer from 0 to 23. Midnight is 0 and 11 PM is 23.
minutes	A minute, specified as an integer from 0 to 59. The first minute of an hour is 0 and the last minute of an hour is 59.
seconds	A second, specified as an integer from 0 to 59. The first second of a minute is 0 and the last second of a minute is 59.
millisecond	A millisecond, specified as an integer from 0 through 999. The first millisecond is 0 and the last millisecond is 999.

About the Date Constructor

The Date constructor instantiates a new date object. If you include an argument, then it returns a date object that includes the date according to the argument. To create a date object that Siebel CRM sets to the current date and time, you can use the new operator as you would with any object.

Format A

```
var dateVar = new Date;
```

Format B

```
var dateVar = new Date(milliseconds);
```

Format C

```
var dateVar = new Date(dateString);
```

Format D

```
var dateVar = new Date(year, month, day);
```

Format E

```
var dateVar = new Date(year, month, day, hours, minutes, seconds);
```

Arguments

Table 62 describes arguments for the date constructor.

Table 62. Arguments for the Date Constructor

Argument	Description
dateString	A string that includes a date and optional time.
year	A year. If the year is between 1950 and 2050, then you can include only the final two digits. Otherwise, you must include four digits. For more information, see "Caution About Using Two Digit Dates" on page 122.
month	For more information, see "Values for Dates and Times" on page 123.
day	
hours	
minutes	
seconds	
milliseconds	The number of milliseconds since January 1, 1970.

Usage for Format B

Format B returns a date and time that includes the number of milliseconds since midnight, January 1, 1970. Using milliseconds is a standard way of including dates and times. It simplifies calculating the amount of time between one date and another. It is recommended that you configure Siebel CRM to convert a date to milliseconds before it performs a calculation on the date.

Usage for Format C

Format C accepts a string that includes a date and an optional time. The format for this string includes one or more of the following fields, in any order:

```
month day, year hours:minutes:seconds
```

For example:

```
"October 13, 1995 13:13:15"
```

This string specifies a date of October 13, 1995 and a time of one thirteen and 15 seconds PM. In a 24 hour format, this value is 13:13 hours and 15 seconds. The time specification is optional. If you include it, then the seconds specification is optional.

Siebel CRM can pass the result of the BusComp.GetFieldValue(*datetime field*) method to the date constructor. The GetFieldValue method always returns date fields using the following format:

```
MM/DD/YYYY hh:mm:ss
```

Siebel CRM interprets the time in a date string as local time, according to the time zone setting of the operating system. If you require Siebel CRM to interpret the time as UTC time, then you can append GMT to the date string. For example:

```
"07/09/2004 14:22:00 GMT"
```

If a business component field includes a UTC time rather than a local time, then you can append GMT to the code to configure Siebel CRM to pass is to the date constructor. For example:

```
var utctime = new Date(GetFieldValue("UTC Time") + " GMT");
```

Usage for Format D and E

Format for formats D and E are self-explanatory. You configure Siebel CRM to pass arguments to them as integers.

Example

The following example includes a date constructor:

```
var aDate = new Date(1802, 6, 23)
```

This example creates a date object that contains a date of July 23, 1802.

Convert Date and Time to String Method

The Convert Date and Time to String method returns a string that includes the date and time of a date object according to the time zone of the computer that runs the script. It returns this date in the following format:

Day Mon dd hh:mm:ss yyyy

If you use this code in Siebel eScript, then the code runs on the Siebel Server. The Siebel Server might or might not reside in the same time zone where the user resides. If you use this code in JavaScript, then the code runs on the user computer and uses the time zone of the user computer.

Format

dateVar.toLocaleString()
dateVar.toString()

Example

The following example displays the local time from the computer clock, the UTC time, and the Greenwich mean time (GMT):

```
var aDate = new Date();
var local = aDate.toLocaleString();
var universal = aDate.toUTCString();
var greenwich = aDate.toGMTString();
TheApplication().RaiseErrorText("Local date is " + local +
   "\nUTC date is " + universal +
   "\nGMT date is " + greenwich);
```

This example provides the following results:

```
Local date is Fri Aug 12 15:45:52 2005
UTC date is Fri Aug 12 23:45:52 2005 GMT
GMT date is Fri Aug 12 23:45:52 2005 GMT
```

Related Topics

For more information, see the following topics:

- "Clib Get Date and Time Method" on page 276
- "Clib Convert Integer to GMT Method" on page 272
- "Clib Convert Integer to Local Time Method" on page 273

Convert Date to Integer Method

The Convert Date to Integer method converts a date object to a system time format that is in the same format as the format that the Clib Convert Time to Integer method returns. To create a date object from a variable in system time format, see "Get Day of Week Method" on page 129.

Format

```
Date.toSystem()
```

Example

The following example converts a date object to a system format that methods of the Clib object can use:

```
var SysDate = objDate.toSystem();
```

Convert Date String to Date Object Method

The Convert Date String to Date Object method converts a date string to a date object. It returns a date object that includes the date in the dateString argument.

Format

```
Date.parse(dateString)
```

Table 63 describes the arguments for the Convert Date String to Date Object method.

Table 63. Arguments for the Convert Date String to Date Object Method

Argument	Description
dateString	A string that uses the following format: *weekday, Month dd, yyyy hh:mm:ss*

Usage

To call the Convert Date String to Date Object method, you use the date constructor rather than a variable. You must use the following format:

```
Friday, October 31, 1998 15:30:00 -0800
```

where:

The last number in the string is the offset from Greenwich mean time.

The following items use this format:

- The *dateVar*.toGMTString method
- Email applications
- Internet applications

You can omit the day of the week, time zone, time specification, and seconds field. For example, consider the following code:

```
var aDate = Date.parse(dateString);
```

This code is equivalent to the following code:

```
var aDate = new Date(dateString);
```

Example

The following example results in a value of 9098766000:

```
var aDate = Date.parse("Friday, October 31, 1998 15:30:00 -0220");
TheApplication().RaiseErrorText(aDate);
```

Convert Date to GMT String Method

The Convert Date to GMT String method converts a date object to a string according to Greenwich mean time. It returns the date that Siebel CRM sets in dateVar. It returns this date as a string in the following format:

Day Mon dd hh:mm:ss yyyy GMT.

Format

dateVar.toGMTString()

Example

The following example accepts a number of milliseconds as input and converts it to GMT time as the number of milliseconds before or after the time on the computer clock:

```
function clickme_Click ()
{
   var aDate = new Date;
   var milli = 200000;
```

```
      aDate.setUTCMilliseconds(milli);
      TheApplication().RaiseErrorText(aDate.toGMTString());
   }
```

Related Topics

For more information, see the following topics:

- "Clib Get Date and Time Method" on page 276
- "Convert UTC Date to Readable Date Method" on page 140

Convert Integer Date to JavaScript Date Method

The Convert Integer Date to JavaScript Date method converts a time from the format that the Clib Convert Time to Integer method returns to a standard JavaScript date object. To call the Convert Integer Date to JavaScript Date method, you use the date constructor rather than a variable.

Format

`Date.fromSystem(`*`time`*`)`

Table 64 describes the arguments for the Convert Integer Date to JavaScript Date method.

Table 64. Arguments for the Convert Integer Date to JavaScript Date Method

Argument	Description
time	A variable that holds a system date.

Example

The following example creates a date object from date information obtained through Clib:

```
var SysDate = Clib.time();
var ObjDate = Date.fromSystem(SysDate);
```

Related Topics

For more information, see the following topics:

- "Clib Create Temporary File Name Method" on page 219
- "About the Date Constructor" on page 123
- "Convert Date to Integer Method" on page 126

Get Day of Month Method

The Get Day of Month method returns the day of the month of a date object. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getDate()

Example

The following example returns a value of 7, the day part of the date object:

```
function Button2_Click ()
{
   var MyBirthdayDay = new Date("1958", "11", "7");
   TheApplication().RaiseErrorText("My birthday is on day " +
      MyBirthdayDay.getDate() + ".");
}
```

Get Day of Week Method

The Get Day of Week method returns the day of the week of a date object as a number from 0 through 6. Sunday is 0 and Saturday is 6.

Format

dateVar.getDay()

Example

To get the name of the corresponding weekday, you can create an array that contains the names of the days of the week, and then compare the return value to the array index. The following example gets the day of the week when New Year's Day occurs:

```
function Button1_Click ()
{
   var weekDay = new Array("Sunday", "Monday", "Tuesday",
      "Wednesday", "Thursday", "Friday", "Saturday");
   var NewYearsDay = new Date("2004", "1", "1");
   var theYear = NewYearsDay.getFullYear()
   var i = 0;
   while (i < NewYearsDay.getDay())
   {
      i++;
      var result = weekDay[i];
   }
   TheApplication().RaiseErrorText("New Year's Day falls on " + result + " in " +
theYear + ".");
}
```

This example displays the following text:

```
New Year's Day falls on Thursday in 2004.
```

Get Full Year Method

The Get Full Year method returns the year of a date object as a number with four digits.

Format

dateVar.getFullYear()

Example

For examples, see the following topics:

- "Get Day of Week Method" on page 129
- "Set Milliseconds Method" on page 135
- "Set Time Method" on page 137

Get Hours Method

The Get Hours method returns the hour of a date object. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getHours()

Example

The following example returns the number 12, which is the hours portion of the specified time:

```
var aDate = new Date("October 31, 1986 12:13:14");
TheApplication().RaiseErrorText(aDate.getHours());
```

Get Milliseconds Method

The Get Milliseconds method returns the milliseconds part of a date object as a number from 0 through 999. When given a date in milliseconds, it returns the last three digits of the millisecond date. If this value is negative, then it returns the result of the last three digits subtracted from 1000. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getMilliseconds()

Example

The following example gets the time from the system clock. The number of milliseconds past the beginning of the second occurs at the end of the message:

```
var aDate = new Date;
   TheApplication().RaiseErrorText( aDate.toString() + " " +
      aDate.getMilliseconds() );
```

Get Minutes Method

The Get Minutes method returns the minutes portion of a date object. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getMinutes()

Example

The following example returns the number 13, which is the minutes portion of the specified time:

```
var aDate = new Date("October 31, 1986 12:13:14");
TheApplication().RaiseErrorText(aDate.getMinutes());
```

Get Month Method

The Get Month method returns the month of a date object. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getMonth()

Example

The following example returns the number 10, with the result of adding 1 to the month portion of the specified date:

```
var aDate = new Date("October 31, 1986 12:13:14");
TheApplication().RaiseErrorText(aDate.getMonth() + 1);
```

Get Seconds Method

The Get Seconds method returns the seconds portion of a date object as a number from 0 through 59. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getSeconds()

Example

The following code returns the number 14, which is the seconds portion of the specified date:

```
var aDate = new Date("October 31, 1986 12:13:14");
TheApplication().RaiseErrorText(aDate.getSeconds());
```

Get Time Method

The Get Time method returns the number of milliseconds for a date object. It returns this value as an integer. This integer includes the number of seconds between midnight on January 1, 1970, GMT, and the date and time that the date object specifies.

Format

dateVar.getTime()

Example

The following example returns a value of 245594000. To convert this value to a value that a person can interpret, you can use the Convert Date and Time to String method or the Convert Date to GMT String Method method:

```
var aDate = new Date("January 3, 1970 12:13:14");
TheApplication().RaiseErrorText(aDate.getTime());
```

Related Topics

For more information, see the following topics:

- “Convert Date and Time to String Method” on page 125
- “Convert Date to GMT String Method” on page 127
- “Clib Get Date and Time Method” on page 276
- “Clib Convert Integer to GMT Method” on page 272
- “Clib Convert Integer to Local Time Method” on page 273
- “Clib Convert Time to Integer Method” on page 274

Get Time Zone Offset Method

The Get Time Zone Offset method returns the difference, in minutes, between UTC time and local time that it calculates as the UTC time minus the local time. For example, Central European Time (CET) is UTC plus 60. On a computer that is set to the CET time zone, the Get Time Zone Offset method returns a value of negative 60.

Format

dateVar.getTimezoneOffset()

Example

The following example calculates the difference from UTC, in hours, of your location, according to the setting in the Windows Control Panel:

```
var aDate = new Date();
var hourDifference = Math.round(aDate.getTimezoneOffset() / 60);
TheApplication().RaiseErrorText("Your time zone is " +
      hourDifference + " hours from GMT.");
```

Get Year Method

The Get Year method returns the year portion of a date object as the offset from a base year of 1900. The offset is positive for any year that occurs after 1900 and is negative for any year that occurs before 1900. For example, if the value of *dateVar* is a date in the year 2004, then *dateVar*.getYear equals 104.

Format

dateVar.getYear()

Set Date Method

The Set Date method sets the day of dateVar to the value you specify in the dayOfMonth argument.

Format

*dateVar.*setDate(*dayOfMonth*)

Table 65 describes the arguments for the Set Date method.

Table 65. Arguments for the Set Date Method

Argument	Description
dayOfMonth	The day of the month to set in dateVar as an integer from 1 through 31. For more information, see “Values for Dates and Times” on page 123.

Setting the Day to a Value That Exceeds 31

You can add any number of days to a date. Siebel eScript automatically converts the number of days to the correct month and year. For example, to add the number of days to a date, you can use the following script:

```
//script to add 7 days to a date
```

```
  var dtNextWeek = new Date();
  dtNextWeek.setDate( dtNextWeek.getDate()+7 );
//script to add 76 days to a date
  var dtNextWeek = new Date();
  dtNextWeek.setDate(dtNextWeek.getDate()+76);
```

Set Full Year Method

The Set Full Year method sets the year of a date object to a four digit year. Optionally, you can use Siebel eScript to set the month of the year argument to the month argument, and the date of the month argument to the date argument. You must express the year in four digits.

Format

dateVar.setFullYear(*year*[, *month*[, *date*]])

Table 66 describes the arguments for the Set Full Year method.

Table 66. Arguments for the Set Full Year Method

Argument	Description
year	The year to set in dateVar as a four digit integer.
month	The month to set in dateVar as an integer from 0 through 11. For more information, see "Values for Dates and Times" on page 123.
date	The date to set in dateVar as an integer from 1 through 31.

Set Hours Method

The Set Hours method sets the hour of a date object to an hour of a 24-hour clock. You can optionally set the UTC minute, second, and millisecond. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.setHours(*hour*[, *minute*[, *second*[, *millisecond*]]])

Table 67 describes the arguments for the Set Hours method.

Table 67. Arguments for the Set Hours Method

Argument	Description
hour	For more information, see "Values for Dates and Times" on page 123.
minute	
second	
millisecond	

Set Milliseconds Method

The Set Milliseconds method sets the millisecond of a date object to a date expressed in milliseconds relative to the system time. The value of dateVar becomes equivalent to the number of milliseconds from the time on the system clock. You can use a positive number for a later time and a negative number for an earlier time.

Format

*dateVar.*setMilliseconds(*millisecond*)

Table 68 describes the arguments for the Set Milliseconds method.

Table 68. Arguments of the Set Milliseconds Method

Argument	Description
millisecond	For more information, see "Values for Dates and Times" on page 123.

Example

The following example accepts a number of milliseconds as input and converts it to the date relative to the date and time in the computer clock:

```
function test2_Click ()
{
   var aDate = new Date;
   var milli = 7200000;
   aDate.setMilliseconds(milli);
   var aYear = aDate.getFullYear();
   var aMonth = aDate.getMonth() + 1;
   var aDay = aDate.getDate();
   var anHour = aDate.getHours();

   switch(anHour)
   {
      case 0:
         anHour = " 12 midnight.";
```

```
            break;
         case 12:
            anHour = " 12 noon.";
            break;
         default:
            if (anHour > 11 )
               anHour = (anHour - 12 ) + " P.M.";
            else
               anHour = anHour + " A.M.";
      }

      TheApplication().RaiseErrorText("The specified date is " + aMonth + "/" + aDay +
   "/" + aYear + " at " + anHour);
   }
```

The number 7200000 milliseconds is two hours. If you run this code on November 22, 2005 between 3 P.M. and 4 P.M., then it provides the following result:

```
The specified date is 11/22/2005 at 5 P.M.
```

Set Minutes Method

The Set Minutes method sets the minute of dateVar to the value you specify in the minute argument. You can optionally set the minute argument to a specific second and millisecond. For more information, see "Values for Dates and Times" on page 123.

Format

*dateVar.*setMinutes(*minute*[, *second*[, *millisecond*]])

Table 69 describes the arguments for the Set Minutes method.

Table 69. Arguments for the Set Minutes Method

Argument	Description
minute	For more information, see "Values for Dates and Times" on page 123.
second	
millisecond	

Set Month Method

The Set Month method sets the month of dateVar to the value you specify in the month argument. You can optionally set the day of month to the date argument. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.setMonth(*month*[, *date*])

Table 70 describes the arguments for the Set Month method.

Table 70. Arguments of the Set Month Method

Argument	Description
month	The month to set in dateVar as an integer from 0 through 11.
date	The date of the month argument to set in dateVar as an integer from 1 through 31.

Set Seconds Method

The Set Seconds method sets the second of dateVar to the value you specify in the second argument. You can optionally use this method to set the second argument to the value that you specify in the millisecond argument.

Format

dateVar.setSeconds(*second*[, *millisecond*])

Table 71 describes the arguments for the Set Seconds method.

Table 71. Arguments for the Set Seconds Method

Argument	Description
second	For more information, see "Values for Dates and Times" on page 123.
millisecond	

Set Time Method

The Set Time method sets dateVar to a date that Siebel CRM determines from the value you specify in the milliseconds argument, calculated from January 1, 1970, GMT. To set a date earlier than this date, you can use a negative number.

Format

*dateVar.*setTime(*milliseconds*)

For more information about the milliseconds argument, see "Values for Dates and Times" on page 123.

Example

The following example uses a number of milliseconds as input and converts it to a date and hour:

```
function dateBtn_Click ()
{
   var aDate = new Date;
   var milli = -4000;
   aDate.setTime(milli);
   var aYear = aDate.getFullYear();
   var aMonth = aDate.getMonth() + 1;
   var aDay = aDate.getDate();
   var anHour = aDate.getHours();

   switch(anHour)
   {
      case 0:
         anHour = " 12 midnight.";
         break;
      case 12:
         anHour = " 12 noon.";
         break;
      default:
         if ( anHour > 11 )
            anHour = (anHour - 12) + " P.M.";
         else
            anHour = anHour + " A.M.";
   }

   TheApplication().RaiseErrorText("The specified date is " +
      aMonth + "/" + aDay + "/" + aYear + " at " + anHour);
}
```

For example, if you enter a value of -345650, then this code provides the following result:

```
The specified date is 12/31/1969 at 3 P.M.
```

Set Year Method

The Set Year method sets the year of a date object as a two digit or four digit year that you specify.

Format

dateVar.setYear(*year*)

Table 72 describes the arguments for the Set Year method.

Table 72. Arguments for the Set Year Method

Argument	Description
year	The year to set in dateVar. You can write code that uses one of the following values in the year argument: ■ A two digit integer for a year that occurs in the twentieth century ■ A four digit integer for a year that does not occur in the twentieth century

UTC Methods

This topic describes UTC methods. It includes the following topics:

Convert UTC Date to Readable Date Method

The Convert UTC Date to Readable Date returns a string that includes the UTC date of dateVar in a format that a human can read. This string uses the following format:

Day Mon dd hh:mm:ss yyyy

Format

dateVar.toUTCString()

Example

For an example, see "Convert Date and Time to String Method" on page 125.

Related Topics

For more information, see the following topics:

- "Clib Get Date and Time Method" on page 276
- "Convert Date to GMT String Method" on page 127
- "Convert Date and Time to String Method" on page 125

Get UTC Date Method

The Get UTC Date method returns an integer that includes the number of milliseconds before or after midnight January 1, 1970 of the date and time that you specify. To call this method, you use the date constructor rather than a variable. This method interprets the arguments as referring to GMT time. For more information, see "Values for Dates and Times" on page 123.

Format

Date.UTC(*year, month, day,* [*, hours*[*, minutes*[*, seconds*]]])

Table 73 describes the arguments for the Get UTC Date method.

Table 73. Arguments for the Get UTC Date Method

Argument	Description
year	An integer that contains the year. To represent a year that occurs in the twentieth century, you can use two digits. For more information, see "Caution About Using Two Digit Dates" on page 122.

Table 73. Arguments for the Get UTC Date Method

Argument	Description
month	For more information, see “Values for Dates and Times” on page 123.
day	
hours	
minutes	
seconds	

Example

The following example uses the Get UTC Date method:

```
function clickme_Click ()
{
   var aDate = new Date(Date.UTC(2005, 1, 22, 10, 11, 12));
   TheApplication().RaiseErrorText("The specified date is " +
      aDate.toUTCString());
}
```

This example provides the following result:

```
The specified date is Sat Jan 22 10:11:12 2005 GMT
```

Related Topics

For more information, see “About the Date Constructor” on page 123.

Get UTC Day of Month Method

The Get UTC Day of Month method returns the UTC day of the month of dateVar as a number from 1 to 31. For more information, see “Values for Dates and Times” on page 123.

Format

dateVar.getUTCDate()

Example

The following example displays 1, the hour portion of the date, followed by the GMT equivalent, which can include the same value:

```
var aDate = new Date("May 1, 2005 13:24:35");
TheApplication().RaiseErrorText("Local day of the month is " +
   aDate.getHours() +"\nGMT day of the month is " +
   aDate.getUTCHours());
```

Get UTC Day of Week Method

The Get UTC Day of Week method returns the UTC day of the week of a date object as a number from 0 through 6. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getUTCDay()

Example

The following example displays the day of the week for May 1, 2005 in local time and in UTC time:

```
function Button2_Click ()
{
   var localDay;
   var UTCDay;
   var MayDay = new Date("May 1, 2005 13:30:35");
   var weekDay = new Array("Sunday", "Monday", "Tuesday",
      "Wednesday", "Thursday", "Friday", "Saturday");

   for (var i = 0; i <= MayDay.getDay();i++)
      localDay = weekDay[i];
   var msgtext = "May 1, 2005, 1:30 PM falls on " + localDay;

   for  (var j = 0; j <= MayDay.getUTCDay(); j++)
      UTCDay = weekDay[j];
   msgtext = msgtext + " locally, \nand on " + UTCDay + " GMT.";

   TheApplication().RaiseErrorText(msgtext);
}
```

Get UTC Full Year Method

The Get UTC Full Year year method returns the UTC year of a date object as a four digit number.

Format

dateVar.getUTCFullYear()

Example

The following example displays 2005, the year portion of the date, followed by the GMT equivalent, which can include the same value:

```
var aDate = new Date("January 1, 2005 13:24:35");
TheApplication().RaiseErrorText("Local year is " + aDate.getYear() +
   "\nGMT year is " + aDate.getUTCFullYear());
```

Get UTC Hours Method

The Get UTC Hours Method returns the UTC hour of a date object as a number from 0 through 23. For more information, see “Values for Dates and Times” on page 123.

Format

dateVar.getUTCHours()

Example

The following example displays a value of 13, which is the hour portion of the date, followed by the GMT equivalent:

```
var aDate = new Date("May 1, 2005 13:24:35");
TheApplication().RaiseErrorText("Local hour is “ + aDate.getHours() +
   "\nGMT hour is " + aDate.getUTCHours());
```

Get UTC Milliseconds Method

The Get UTC Milliseconds method returns the UTC millisecond of a date object as a number from 0 through 999. For more information, see “Values for Dates and Times” on page 123.

Format

dateVar.getUTCMilliseconds()

Get UTC Minutes Method

The Get UTC Minutes method returns the UTC minute of a date object as a number from 0 through 59. For more information, see “Values for Dates and Times” on page 123.

Format

dateVar.getUTCMinutes()

Example

The following example displays a value of 24, which is the minutes portion of the date, followed by the GMT equivalent:

```
var aDate = new Date("May 1, 2005 13:24:35");
TheApplication().RaiseErrorText("Local minutes: " + aDate.getMinutes() +
   "\nGMT minutes: " + aDate.getUTCMinutes());
```

Get UTC Month Method

The Get UTC Month method returns the UTC month of a date object as a number from 0 through 11. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getUTCMonth()

Example

The following example displays a value of 5, which is the month portion of the date determined by adding 1 to the value that the Get UTC Month method returns. This value is followed by the GMT equivalent which is determined by adding 1 to the value that the Get UTC Month method returns:

```
var aDate = new Date("May 1, 2005 13:24:35");
var locMo = aDate.getMonth() + 1;
var GMTMo = aDate.getUTCMonth() + 1
TheApplication().RaiseErrorText("Local month: " + locMo +"\nGMT month: "
   + GMTMo);
```

Get UTC Seconds Method

The Get UTC Seconds method returns the UTC second of a date object as number from 0 through 59. For more information, see "Values for Dates and Times" on page 123.

Format

dateVar.getUTCSeconds()

Set UTC Date Method

The Set UTC Date method sets the UTC day of a date object to a number from 1 through 31 according to the value you set in the dayOfMonth argument.

Format

dateVar.setUTCDate(*dayOfMonth*)

Table 74 describes the arguments for the Set UTC Date method.

Table 74. Arguments for the Set UTC Date Method

Argument	Description
dayOfMonth	The day of the UTC month to set in dateVar as an integer from 1 through 31. For more information, see "Values for Dates and Times" on page 123.

Set UTC Full Year Method

The Set UTC Full Year method sets the UTC year of a date object to a four digit year that you specify in the year argument.

Format

dateVar.setUTCFullYear(*year*[, *month*[, *date*]])

Table 75 describes the arguments for the Set UTC Full Year method.

Table 75. Arguments for the Set UTC Full Year Method

Argument	Description
year	The UTC year to set in dateVar as a four digit integer. You must express the year in four digits.
month	As an option, you can use the month argument and the date argument to set the month and date of the year. For more information, see "Values for Dates and Times" on page 123.
date	

Example

The following example does the following work:

- To assign the date of the 2000 summer solstice, it uses the Set UTC Full Year method
- To assign time to a date object, it uses the Set UTC Hours method
- Determines the local date and displays it

This example uses the following code:

```
function dateBtn_Click ()
{
   var Mstring = " A.M., Standard Time.";
   var solstice2K = new Date;
   solstice2K.setUTCFullYear(2000, 5, 21);
   solstice2K.setUTCHours(01, 48);
   var localDate = solstice2K.toLocaleString();
   var pos = localDate.indexOf("2000")
   var localDay = localDate.substring(0, pos - 10);

   var localHr = solstice2K.getHours();
   if (localHr > 11 )
   {
      localHr = (localHr - 12 );
      Mstring =  " P.M., Standard Time.";
   }
   var localMin = solstice2K.getMinutes();
```

```
      var msg = "In your location, the solstice is on " + localDay +
         ", at " + localHr + ":" + localMin + Mstring;
      TheApplication().RaiseErrorText(msg);
   }
```

This example produces the following result:

```
In your location, the solstice is on Tue Jun 20, at 6:48 P.M., Standard Time.
```

Set UTC Hours Method

The Set UTC Hours method sets the UTC hour of a date object to a specific hour of a 24-hour clock as a number from 0 through 23. As an option, you can also set the UTC minute, second, and millisecond.

Format

dateVar.setUTCHours(*hour*[, *minute*[, *second*[, *millisecond*]]])

Table 76 describes the arguments for the Set UTC Hours method.

Table 76. Arguments for the Set UTC Hours Method

Argument	Description
hour	For more information, see "Values for Dates and Times" on page 123.
minute	
second	
millisecond	

Example

For an example, see "Set UTC Full Year Method" on page 145.

Set UTC Milliseconds Method

The Set UTC Milliseconds method sets the UTC millisecond of a date object to a date expressed in milliseconds relative to the UTC equivalent of the system time. The value of dateVar becomes equivalent to the number of milliseconds from the UTC equivalent of the time on the system clock. You can use a positive number for later times or a negative number for earlier times.

Format

*dateVar.*setUTCMilliseconds(*millisecond*)

Table 77 describes the arguments for the Set UTC Milliseconds method.

Table 77. Arguments for the Set UTC Milliseconds Method

Argument	Description
millisecond	The UTC millisecond to set in dateVar as a positive or negative integer. For more information, see "Values for Dates and Times" on page 123.

Example

The following example gets a number of milliseconds as input and converts it to a UTC date and time:

```
function dateBtn_Click ()
{
   var aDate = new Date;
   var milli = 20000;
   aDate.setUTCMilliseconds(milli);
   var aYear = aDate.getUTCFullYear();
   var aMonth = aDate.getMonth() + 1;
   var aDay = aDate.getUTCDate();
   var anHour = aDate.getUTCHours();
   var aMinute = aDate.getUTCMinutes();
   TheApplication().RaiseErrorText("The specified date is " +
            aMonth +
      "/" + aDay + "/" + aYear + " at " + anHour + ":" +
      aMinute + ", UTC time.");
}
```

If run at 5:36 P.M., PST (Pacific Standard Time), on August 22, 2005, then this example produced the following result:

```
The specified date is 8/23/2005 at 1:36 UTC time.
```

Set UTC Minutes Method

The Set UTC Minutes method sets the UTC minute of a date object to a minute that you specify in the minute argument.

Format

*dateVar.*setUTCMinutes(*minute*[, *second*[, *millisecond*]])

Table 78 describes the arguments for the Set UTC Minutes method.

Table 78. Arguments for the Set UTC Minutes Method

Argument	Description
minute	As an option, you can use the second argument to set the minute to a specific UTC second and the millisecond argument to set the minute to a UTC millisecond. For more information, see "Values for Dates and Times" on page 123.
second	
millisecond	

Set UTC Month Method

The Set UTC Month method sets the UTC month of a date object to a specific month.

Format

dateVar.setUTCMonth(*month*[, *date*])

Table 79 describes the arguments for the Set UTC Month method.

Table 79. Arguments for the Set UTC Month Method

Argument	Description
month	The UTC month to set in dateVar as an integer from 0 through 11. As an option, you can set this argument to the value that the date argument contains. For more information, see "Values for Dates and Times" on page 123.
date	The UTC date of the month argument to set in dateVar as an integer from 1 through 31.

Set UTC Seconds Method

The Set UTC Seconds method sets the UTC second of the minute of a date object to a second that you specify.

Format

dateVar.setUTCSeconds(*second*[, *millisecond*])

Table 80 describes the arguments for the Set UTC Seconds method.

Table 80. Arguments for the Set UTC Seconds Method

Argument	Description
second	As an option, you can set the second argument to a value that you specify in the millisecond argument. For more information, see "Values for Dates and Times" on page 123.
millisecond	

Global Methods

This topic describes global methods. It includes the following topics:

Overview of Global Methods

A *global method* is a method of the global object.

A *global variable* is a member of a global object. To reference a global property, you do not need to use an object name. For example, to reference the Is NaN method that tests to determine if a value is equal to the special value NaN, you can use the format that this topic describes. For more information, see "Is NaN Method" on page 176.

The global methods that this book describes are unique to the Siebel eScript implementation of JavaScript. These methods are not part of the ECMAScript standard. Avoid using them in a script that you might use with a JavaScript interpreter that does not support them.

You can use format A or format B to call a global method.

Format A

```
globalMethod(value);
```

Format A treats the globalMethod argument as a function.

You cannot use format A in a function that includes a local variable that has the same name as a global variable. To reference the global variable in this situation, you must use the global keyword.

Format B

```
global.globalMethod(value);
```

Format B treats the globalMethod argument as a method of the global object.

Arguments

Table 81 describes the arguments of a global object.

Table 81. Arguments of a Global Method

Argument	Description
globalMethod	The method that the global object applies.
value	The value that the global object applies to the method that you specify in the globalMethod argument.

Related Topics

For more information, see the following topics:

- "NaN Numbers" on page 28
- "Conversion Methods" on page 154

Create COM Object Method

The Create COM Object method instantiates a COM object. It returns a successful COM object or an undefined object.

Format

`COMCreateObject(`*`objectName`*`)`

Table 82 describes the arguments for the Create COM Object method.

Table 82. Arguments for the Create COM Object Method

Argument	Description
objectName	The name of the object that this method creates.

Usage

You can configure Siebel CRM to pass any type of variable to the COM object that it calls. You must make sure the variable type is valid for the COM object. The following variable types are valid:

- String
- Number
- Object pointer

Siebel CRM can run the Create COM Object method only in server script. It cannot run this method in browser script.

A DLL that the Create COM Object method instantiates must be thread-safe.

Using the Dispatch Identifier to Call a COM Method

Siebel CRM calls the method of a COM object in Siebel eScript in the same way that it calls this method in Siebel VB. In this context, a *COM object* is an object that the Create COM Object method instantiates.

To use the DISPID (Dispatch Identifier) of a COM method to call that COM method, you make an IDispatch::Invoke call in the COM technology. To identify methods, properties, and arguments, you use the Dispatch Identifier in the IDispatch::Invoke call.

You can write code that uses only the following arguments:

- **BSTR (basic string).** An eScript string.
- **VARIANT.** A universal data type.
- **SAFEARRAY.** Similar to a typical C array, but also includes information about the number of elements in the array.

You cannot use Siebel eScript to call the method of a COM object that includes the LPCSTR argument for the string argument of that method. In this situation, you must use the BSTR argument.

Example

The following example instantiates Microsoft Excel as a COM object and makes it visible:

```
var ExcelApp = COMCreateObject("Excel.Application");

// Make Excel visible through the Application object.
ExcelApp.Visible = true;
ExcelApp.WorkBooks.Add();

// Place some text in the first cell of the sheet
ExcelApp.ActiveSheet.Cells(1,1).Value = "Column A, Row 1";

// Save the sheet
var fileName = "C:\\demo.xls";
ExcelApp.ActiveWorkbook.SaveAs (fileName);

// Close Excel with the Quit method on the Application object
ExcelApp.Application.Quit();

// Clear the object from memory
ExcelApp = null;
return (CancelOperation);
```

An application, such as Microsoft Excel, might change from version to version, so it might be necessary for you to modify your code to address these modifications. This example code was tested on Excel 2003.

Get Array Length Method

The Get Array Length method returns the length of a dynamically created array. This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

Note the following:

- You can write code that uses the Get Array Length method only with a dynamically created array. You cannot use it with an array that is not created with the Array constructor and the new operator.
- The length property is not available for a dynamically created array. A dynamically created array must use the Get Array Length method or the Set Array Length method when working with an array length.
- If you work with an array that the array constructor and the new operator creates, then you must use the length property of the array.

For more information, see "Use Caution If You Define an Array That Includes a Negative Index" on page 153.

Format

```
getArrayLength(array[, minIndex])
```

Table 83 describes the arguments for the Get Array Length method.

Table 83. Arguments for the Get Array Length Method

Argument	Description
array	The name of the array whose length this method must get.
minIndex	The index of the lowest element where this method starts counting. The first element of an array is typically at index 0. If you specify the minIndex argument, then Siebel CRM uses it to set to the minimum index, which is zero or less.

Related Topics

For more information, see the following topics:

- "About Array Functions" on page 77
- "Get Largest Array Index Method" on page 82
- "Set Array Length Method" on page 153

Set Array Length Method

The Set Array Length method sets the first index and length of an array. It sets the length of the array argument to a range that the minIndex argument and the length argument define.

If you specify all three arguments for this method, then the following occurs:

- The minIndex argument is the minimum index of the resized array.
- The length argument is the length of the resized array.
- If an element resides outside the length of the resized array, then that element becomes undefined.

If you only specify two arguments, then this method uses the second argument as the length argument and sets the minimum index of the resized array to 0 by default.

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

Format

```
setArrayLength(array[, minIndex], length])
```

Table 84 describes the arguments for the Set Array Length method.

Table 84. Arguments for the Set Array Length Method

Argument	Description
array	The name of the array whose length this method must set.
minIndex	The index of the lowest element where this method starts counting. This value must be 0 or less. If you use ST eScript code, then it is not appropriate to use the minIndex argument. If you use ST eScript code, then this code restricts the minimum index to zero only and assigns it by default.
length	The length of the array.

Use Caution If You Define an Array That Includes a Negative Index

Use caution if you defined an array that includes a negative index.

CAUTION: ST eScript code does not support a negative array index. If you define an array that includes a negative index, and if you use T eScript code to define this array in a Siebel application prior to release 7.8, then you must redefine the index range for this array and any references according to index values. As an alternative to using the Set Array Length method to set the array length, you can use the length property of the array object.

Related Topics

For more information, see the following topics

- "Get Array Length Method" on page 152
- "Get Largest Array Index Method" on page 82

Undefine Method

The Undefine method undefines a variable, object property, or value. Assume Siebel CRM defines a value, and then a defined method returns true for this value. If you use the Undefine method with this value, then the Is Defined method returns false. Undefining a value is not the same as setting a value to null.

The following example sets the n variable to 2, and then undefines the n variable:

```
var n = 2;
undefine(n);
```

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

Format

undefine(*value*)

Table 85 describes the arguments for the Undefine method.

Table 85. Arguments for the Undefine Method

Argument	Description
value	The variable or object property that this method must undefine.

Example

The following example creates an object named o, and then defines an o.one property. It then undefines this property but the o object remains defined:

```
var o = new Object;
o.one = 1;
undefine(o.one);
```

Conversion Methods

This topic describes conversion methods. It includes the following topics:

- "Overview of Conversion Methods" on page 155
- "Convert String to Floating-Point Number Method" on page 156
- "Convert String to Integer Method" on page 157
- "Convert Number to Exponential Notation Method" on page 158

Overview of Conversion Methods

You might encounter a situation where you must specify or control the types of variables or data. Some conversion methods include one argument that is a variable or data item that Siebel eScript converts to the data type that you specify in the name of the method. For example, the following code creates two variables:

```
var aString = ToString(123);
var aNumber = ToNumber("123");
```

In this example, Siebel eScript does the following work:

- To create the aString variable, it converts the number 123 to a string.
- To create the aNumber variable, it converts the string value "123" to a number.

It already created the aString variable with a value of "123", so the second code line can use the following format:

```
var aNumber = ToNumber(aString);
```

Convert String to Floating-Point Number Method

The Convert String to Floating-Point Number method converts an alphanumeric string to a floating-point decimal number. It returns a floating-point decimal number. If it cannot convert to a number the value that the string argument contains, then it returns the following value:

NaN

For more information, see "NaN Numbers" on page 28.

Format

```
parseFloat(string)
```

Table 86 describes the arguments for the Convert String to Floating-Point Number method.

Table 86. Arguments for the Convert String to Floating-Point Number Method

Argument	Description
string	The string that this method must convert.

How the Convert String to Floating-Point Number Method Handles the String

The first character that is not a white space character must be a digit or a minus sign (-). For more information, see "Use White Space to Improve Readability" on page 56.

The Convert String to Floating-Point Number method does the following:

- Ignores white space characters that occur at the beginning of the string
- Treats the first period (.) in the string as a decimal point
- Treats any digits that follow the first period as the fractional part of the number
- Stops reading the string at the first nonnumeric character that occurs after the decimal point
- Ignores the first nonnumeric character it encounters
- Ignores all characters that occur after the first nonnumeric character
- Converts the result into a number

Example

The following example returns a result of negative 234.37:

```
var num = parseFloat(" -234.37 profit");
```

Convert String to Integer Method

The Convert String to Integer method converts an alphanumeric string to an integer. It returns an integer. If it cannot convert the value that the string argument contains to a number, then it returns the following value:

NaN

For more information, see "NaN Numbers" on page 28.

Format

parseInt(*string [,radix]*)

Table 87 describes the arguments for the Convert String to Integer method.

Table 87. Arguments for the Convert String to Integer Method

Argument	Description
string	The string that this method converts.
radix	The base of the number system that this method uses in the return value. For example, if you set the radix argument to 8, then it returns the value as an octal number.

Usage

If you do not specify the radix argument or if the value that the radix argument contains is zero, then the Convert String to Integer method uses a value of 10 for the radix unless the value that the string argument contains begins with one of the following values:

- **The character pairs 0x or 0X.** It uses a value of 16 for the radix.
- **A zero and a valid octal digit.** It uses a value of 8 for the radix. Any number zero through seven is a valid octal digit.

CAUTION: If the passed string includes a leading zero, such as 05, then the Convert String to Integer method interprets the number as on octal. An argument that it interprets as an invalid octal creates a return value of zero. The values 08 and 09 are examples of invalid octal values.

This method handles the string in the same way as the Convert String to Floating-Point Number method. For more information, see "How the Convert String to Floating-Point Number Method Handles the String" on page 156.

Example

The following example returns a result of negative 234:

```
var num = parseInt(" -234.37 profit");
```

Convert Number to Exponential Notation Method

The Convert Number to Exponential Notation method converts a number to exponential notation. It returns the number that the numberVar variable contains, expressed in exponential notation to the number of decimal places that you specify in the len argument.

Format

numberVar.toExponential(*len*)

Table 88 describes the arguments for the Convert Number to Exponential Notation method.

Table 88. Arguments for the Convert Number to Exponential Notation Method

Argument	Description
len	The number of decimal places in the significant digits portion of the number.

How the Convert Number to Exponential Notation Method Handles the Len Argument

The Convert Number to Exponential Notation method does one of the following depending on one of the following values that the len argument contains:

- Less than the number of significant decimal places that the numberVar variable contains. It does one of the following:
  - If the number is five or greater, then it rounds the result up.
  - If the number is less than five, then it rounds the result down.
- Greater than the number of significant decimal places that the numberVar variable contains. It pads the extra places with zeroes.
- Negative. It creates an error.

Using a Multivalue List to Avoid Unexpected Rounding

If you must use a value that exceeds 2^{53}, then it is recommended that you use a calculated field that uses the sum of a multivalue list instead of using Siebel eScript. If Siebel CRM performs an operation that results in a value that exceeds 2^{53}, then it rounds this value to 2^{53}.

The largest number that the Siebel eScript engine can hold is 2^{53}. This number is equivalent to the following values:

- $9.00719925 \times 10^{15}$, with rounding
- 9,007,199,254,740,992, without rounding

Example

The following example uses the Convert Number to Exponential Notation method:

```
var num = 1234.567
var num3 = num.toExponential(3) //returns 1.235e+3
var num2 = num.toExponential(0) //returns 1e+3
var num9 = num.toExponential(9) //returns 1.234567000e+3

var smallnum = 0.0001234
var smallnum2 = smallnum.toExponential(2) //returns 1.2e-4
var smallnumerr = smallnum.toExponential(-1) //throws error
```

Convert Number to Fixed Decimal Method

The Convert Number to Fixed Decimal method converts a number according to the decimal places that you specify. It returns the number that it converts. It allows you to express a number that includes a number of decimal places that you specify. For example, to express the results of a currency calculation that includes two decimal places.

This method does the same work as the Convert Number to Exponential Notation method. For more information, see "How the Convert Number to Exponential Notation Method Handles the Len Argument" on page 158.

This method uses the same argument as the Convert Number to Exponential Notation method. For more information, see Table 88 on page 158.

Format

numberVar.toFixed(*len*)

Example

The following example uses the Convert Number to Fixed Decimal method:

```
var profits=2487.8235
var profits3 = profits.toFixed(3) //returns 2487.824
var profits2 = profits.toFixed(2) //returns 2487.82
var profits7 = profits.toFixed(7) //returns 2487.8235000
var profits0 = profits.toFixed(0) //returns 2488
var profitserr = profits.toFixed(-1) //throws error
```

Convert Number to Precision Method

The Convert Number to Precision method converts a number to a number that includes a number of significant digits. It returns the converted number contained in the numberVar variable, expressed to the number of significant digits that you specify in the len argument.

This method allows you to express a number at a desired length. For example, the result of a scientific calculation might only require accuracy to a specific number of significant digits.

This method does one of the following depending on if the value that the len argument contains is:

- Less than the number of significant decimal places that exist in the value that the numberVar variable contains. It does one of the following:

- If the number is five or greater, then it rounds the result up.
- If the number is less than five, then it rounds the result down.

- Greater than the number of significant decimal places that exist in the value that the numberVar variable contains. It pads the extra digits with zeroes and adds a decimal point, if necessary.

This method uses the same argument as the Convert Number to Exponential Notation method. For more information, see Table 88 on page 158.

Format

numberVar.toPrecision(*len*)

Example

The following example uses the Convert Number to Precision method:

```
var anumber = 123.45
var a6 = anumber.toPrecision(6) //returns 123.450
var a4 = anumber.toPrecision(4) //returns 123.5
var a2 = anumber.toPrecision(2) //returns 1.2e+2
```

Convert Special Characters to URL Method

The Convert Special Characters to URL method replaces special characters that a string contains with character combinations so that Siebel CRM can use the string with a URL. It returns a modified string.

Format

escape(*string*)

Table 89 describes the arguments for the Convert Special Characters to URL method.

Table 89. Arguments for the Convert Special Characters to URL Method

Argument	Description
string	A string that contains the characters that this method replaces.

Usage

The character combinations include Unicode values. For a character in the standard ASCII set, this is the hexadecimal ASCII code of the character preceded by a percentage symbol (%). The standard ASCII set includes decimal values 0 through 127.

The following items remain in the string:

- Uppercase letters
- Lowercase letters
- Numbers

- Ampersand (@)
- Asterisk (*)
- Plus sign (+)
- Underscore (_)
- Period (.)
- Forward slash (/)

This method replaces other characters with their respective Unicode sequence.

Example 1

The following example encodes a string. It does not replace the ampersand (@) or asterisk (*) characters:

```
var str = escape("@#$*96!");
```

This example provides the following result:

```
"@%23%24*96%21"
```

Example 2

The following example encodes a string:

```
var encodeStr = escape("@#$*%!");
```

This example provides the following result:

```
"@%23%24*%25%21"
```

Convert Unicode to ASCII Method

The Convert Unicode to ASCII method converts Unicode character combinations that exist in a string to equivalent ASCII characters. It returns the revised string.

Format

unescape(*string*)

Table 90 describes the arguments for the Convert Unicode to ASCII method.

Table 90. Arguments for the Convert Unicode to ASCII Method

Argument	Description
string	A string literal or string variable that contains the Unicode character combinations that this method converts.

Example

The following example displays the string in the argument. The Convert Unicode to ASCII method converts the Unicode character combinations to printable characters. The %20 is the Unicode representation of the space character. The following example normally displays on a single line because a new line cannot break a string:

```
TheApplication().RaiseErrorText(unescape("http://obscushop.com/texis/
%20%20showcat.html?catid=%232029
rg=r133"));
```

This example produces the following result:

```
http://obscushop.com/texis/  showcat.html?catid=#2029
rg=r133
```

Convert Value to Boolean Method

The Convert Value to Boolean method converts a value to the Boolean data type. It returns a value that depends on the data type of the value that the value argument contains. This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

Format

ToBoolean(*value*)

Table 91 describes the arguments for the Convert Value to Boolean method.

Table 91. Arguments for the Convert Value to Boolean Method

Argument	Description
value	The value that this method converts to a Boolean value.

Values That the Convert Value to Boolean Method Returns

Table 92 describes the values that the Convert Value to Boolean method returns.

Table 92. Values That the Convert Value to Boolean Method Returns

Data Type	Return Value
Boolean	Value that the value argument contains.
buffer	This method returns one of the following values depending on if the buffer is empty: ■ **Buffer is empty.** It returns false. ■ **Buffer is not empty.** It returns true.

Table 92. Values That the Convert Value to Boolean Method Returns

Data Type	Return Value
null	False
number	This method returns one of the following values: ■ If the value that the value argument contains is one of the following, then it returns false: ■ 0 ■ +0 ■ -0 ■ NaN ■ If the value that the value argument contains is not 0, +0, -0, or NaN, then it returns false. For more information, see "NaN Numbers" on page 28.
object	True
string	This method returns one of the following values depending on if the string is empty: ■ **The string is empty.** It returns false. ■ **The string is not empty.** It returns true.
undefined	False

Convert Value to Buffer Method

The Convert Value to Buffer method converts the value that the value argument contains to a sequence of ASCII bytes. It then places this value in a buffer. These bytes depend on the data type of the value that the value argument contains. This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

Format

`ToBuffer(`*`value`*`)`

Table 93 describes the arguments for the Convert Value to Buffer method.

Table 93. Arguments for the Convert Value to Buffer Method

Argument	Description
value	The value that this method saves to a buffer.

Values That the Convert Value to Buffer Method Returns

Table 94 describes the values that the Convert Value to Buffer method returns.

Table 94. Values That the Convert Value to Buffer Method Returns

Data Type	Return Value
Boolean	This method returns one of the following values: ■ If the value that the value argument contains is false, then it returns the following value: false ■ If the value that the value argument contains is not false, then it returns the following value: true
null	This returns the following string: null
number	This method returns a value depending on which of the following values the value argument contains: ■ **NaN.** It returns the following value: NaN ■ **+0 or -0.** It returns the following value: 0 ■ **POSITIVE_INFINITY or NEGATIVE_INFINITY.** It returns the following value: Infinity ■ **A number.** It returns a string that includes this number. For more information on the number object, see “NaN Numbers” on page 28.
object	This method returns the following string: [object Object]
string	This method returns the text of the string.
undefined	This method returns the following string: undefined

Convert Value to Bytes Method

The Convert Value to Bytes method converts the value that the value argument contains to bytes, and then places this value in a buffer. This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

This method does not convert a Unicode value to a corresponding ASCII value. For example, it stores the Unicode string Hit as the following value:

\OH\Oi\Ot

This value is the following hexadecimal sequence:

00 48 00 69 00 74

Format

`ToBytes(`*`value`*`)`

Table 95 describes the arguments for the Convert Value to Bytes method.

Table 95. Arguments for the Convert Value to Bytes Method

Argument	Description
value	The value that this method converts to bytes, and then places in a buffer.

Convert Value to Integer Method

The Convert Value to Integer method converts the value that the value argument contains to an integer in the range of negative 2^{15} through 2^{15} minus 1. The equivalent nonexponential range is negative 32,768 through 32,767. It returns a value depending on which of the following values the value argument contains:

- **NaN.** It returns the following value:

  +0

- **+0.** It returns the following value:

  -0

- **POSITIVE_INFINITY or NEGATIVE_INFINITY.** It returns the result.

- **A number.** It rounds the integer part of this number toward zero, and then returns the integer.

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

This method uses the same arguments as the Convert Value to Integer 32 method. For more information, see Table 96 on page 167.

Format

```
ToInteger(value)
```

Usage

To avoid an error, you must first pass the value that the value argument contains to the Is NaN method or to the Convert Value to Number method. To use the Convert Value to Number method, you can include a statement that uses the following format:

```
var x;
x = toNumber(value);
(if x == 'NaN')
.
.   [error -handling statements];
.
else
   ToInteger(value);
```

The Convert Value to Integer method truncates rather than rounds the value it receives. It rounds numbers toward 0. For example, it rounds negative 12.88 to negative 12. It rounds 12.88 to 12.

Related Topics

For more information, see the following topics:

- "NaN Numbers" on page 28
- "Round Number Method" on page 192

Convert Value to Integer 32 Method

The Convert Value to Integer 32 method converts the value that the value argument contains to an integer in the range of negative 2^{31} through 2^{31} minus 1. The equivalent nonexponential range is negative 2,147,483,648 through 2,147,483,647. It returns a value depending on which of the following values the value argument contains:

- **NaN.** It returns the following value:

  NaN

- **+0 or -0.** It returns the following value:

  0

- **POSITIVE_INFINITY or NEGATIVE_INFINITY.** It returns the following value:

  Infinity

- **A number.** It rounds the integer part of this number toward zero, and then returns the integer.

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57 and "NaN Numbers" on page 28.

Format

```
ToInt32(value)
```

Table 96 describes the arguments for the Convert Value to Integer 32 method.

Table 96. Arguments of the Convert Value to Integer 32 Method

Argument	Description
value	The value that this method converts.

Usage

To avoid an error, you must first pass the value that the value argument contains to the Is NaN method or to the Convert Value to Number method. To use the Is NaN method, you include a statement that uses the following format:

```
if (isNaN(value))
.
.   [error-handling statements];
.
else
   ToInt32(value);
```

The Convert Value to Integer 32 method truncates rather than rounds the value it receives, so it rounds numbers toward 0. For example, it rounds negative 12.88 to negative 12. It rounds 12.88 to 12.

Convert Value to Unsigned Integer 16 Method

The Convert Value to Unsigned Integer 16 method converts the value that the value argument contains to an integer in the range of 0 through 2^{16} minus 1. The nonexponential value is 0 through 65,535. It returns a value depending on which of the following values the value argument contains:

- **NaN.** It returns the following value:

  +0

- **+0.** It returns the following value:

  0

- **POSITIVE_INFINITY.** It returns the following value:

  Infinity

- **Any other value.** It returns the absolute value of the integer part of the number, rounded toward 0. The absolute value does not include a positive sign or a negative sign.

This method is unique to Siebel eScript. For more information, see “Make Sure the JavaScript Interpreter Can Run a Function” on page 57.

This method uses the same argument as the Convert Value to Integer 32 method. For more information, see Table 96 on page 167.

Format

`ToUint16(`*`value`*`)`

Usage

To avoid an error, you must first pass the value argument to the Is NaN method or to the Convert Value to Number method. To use the Convert Value to Number method, you can include a statement that uses the following format:

```
var x;i
x = toNumber(value);
(if x == 'NaN')
.
.   [error -handling statements];
.
else
   ToUint16(value);
```

The Convert Value to Unsigned Integer 16 method truncates rather than rounds the value it receives, so it rounds numbers toward 0. For example, it rounds 12.88 to 12.

Related Topics

For more information, see the following topics:

- "NaN Numbers" on page 28
- "Round Number Method" on page 192

Convert Value to Unsigned Integer 32 Method

The Convert Value to Unsigned Integer 32 method converts the value that the value argument contains to an integer in the range of 0 through 2^{32} minus 1. The nonexponential value is 0 through 4,294,967,296. It returns the same value as the Convert Value to Unsigned Integer 16 method. For more information, see "Convert Value to Unsigned Integer 16 Method" on page 167.

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

This method uses the same argument as the Convert Value to Integer 32 method. For more information, see Table 96 on page 167.

Format

`ToUint32(`*`value`*`)`

Usage

To avoid an error, you must first pass the value argument to the Is NaN method or to the Convert Value to Number method. To use the Convert Value to Number method, you can include a statement that uses the following format:

```
if (isNaN(value))
.
.   [error-handling statements];
.
else
   ToUint32(value);
```

The Convert Value to Unsigned Integer 32 method truncates rather than rounds the value it receives, so it rounds numbers toward 0. For example, it rounds 12.88 to 12.

Related Topics

For more information, see the following topics:

- "NaN Numbers" on page 28
- "Round Number Method" on page 192

Convert Value to Number Method

The Convert Value to Number method converts the value that the value argument contains to a number. It returns a value that depends on the original data type of the value that the value argument contains. Table 95 on page 165 describes these data types.

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

This method uses the same argument as the Convert Value to Integer 32 method. For more information, see Table 96 on page 167.

Format

`ToNumber(value)`

Values That the Convert Value to Number Method Returns

Table 97 describes values that the Convert Value to Number method returns.

Table 97. Values That the Convert Value to Number Method Returns

Data Type	Return Value
Boolean	This method returns one of the following values, depending on if the value that the value argument contains is: ■ **False.** It returns the following value: +0 ■ **True.** It returns the following value: 1
buffer string	This method returns one of the following values, depending on if the conversion is: ■ **Successful.** It returns the value that the value argument contains. ■ **Not successful.** It returns the following value: NaN For more information on the number object, see "NaN Numbers" on page 28.
null	0
number	This method returns the value that the value argument contains.
object undefined	NaN

Related Topics

For more information, see "Round Number Method" on page 192.

Convert Value to Object Method

The Convert Value to Object method converts the value that the value argument contains to an object. It returns a value that depends on the data type of the value that the value argument contains.

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

This method uses the same argument as the Convert Value to Integer 32 method. For more information, see Table 96 on page 167.

Format

```
ToObject(value)
```

Data Types of the Value That the Convert Value to Object Method Returns

Table 98 describes data types of the value that the Convert Value to Object method returns.

Table 98. Data Types of the Value That the Convert Value to Object Method Returns

Data Type	Returns
Boolean	A new Boolean object that includes the value that the value argument contains.
number	A new number object that includes the value that the value argument contains.
string	A new string object that includes the value that the value argument contains.
object	The value that the value argument contains.
null	A run-time error.
undefined	

Convert Value to String Method

The Convert Value to String method converts the value that the value argument contains to a string. It returns a value in the format of a Unicode string. The contents of this string depends on the data type of the value that the value argument contains.

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

This method uses the same argument as the Convert Value to Integer 32 method. For more information, see Table 96 on page 167.

Format

```
ToString(value)
```

Values That the Convert Value to String Method Returns

Table 99 describes values that the Convert Value to String method returns.

Table 99. Values That the Convert Value to String Method Returns

Data Type	Return Values
Boolean	This method returns one of the following values, depending on if the value that the value argument contains is: ■ **False.** It returns the following value: false ■ **Not false.** It returns the following value: true
null	This method returns the following string: null
number	This method returns a value depending on which of the following values the value argument contains: ■ **NaN.** It returns the following value: NaN ■ **+0 or -0.** It returns the following value: 0 ■ **Infinity.** It returns the following value: Infinity **A number.** It returns a string that includes this number. For more information on the number object, see “NaN Numbers” on page 28.
object	This method returns the following string: [object Object]
string	This method returns the value that the value argument contains.
undefined	This method returns the following string: undefined

Example

For an example, see “Evaluate Expression Method” on page 173.

Evaluate Expression Method

The Evaluate Expression method evaluates the value that the expression argument contains. It returns the value that it evaluates in the expression argument. If the expression argument is a string, then this method attempts to interpret the string as if it is JavaScript code. If this method:

- **Interprets the string.** It returns the value in the expression argument.
- **Cannot interpret the string.** It returns the following value:

  undefined

If the expression is not a string, then this method returns the value that exists in the expression argument. For example, calling eval(5) returns the value 5.

Format

eval(*expression*)

Table 100 describes the arguments for the Evaluate Expression method.

Table 100. Arguments for the Evaluate Expression Method

Argument	Description
expression	The expression that this method must evaluate.

Example

The following example describes the result of using the Evaluate Expression method on different types of expressions. This method does the following work:

- Interprets the string in the test[0] variable because it can interpret this string as a JavaScript statement.
- Does not interpret the string in the test[1] variable or the test[3] variable because it cannot interpret either string as a JavaScript statement. It returns a value of undefined for each of these variables.

This example includes the following code:

```
function clickme_Click ()
{
   var msgtext = "";
   var a = 7;
   var b = 9;
   var test = new Array(4);
   var test[0] = "a * b";
   var test[1] = ToString(a * b);
   var test[2] = a + b;
   var test[3] = "Strings are undefined.";
   var test[4] = test[1] + test[2];
```

```
for (var i = 0; i < 5; i++)
   msgtext = msgtext + i + ": " + eval(test[i]) + "\n";
TheApplication().RaiseErrorText(msgtext);
```

Running this code produces the following result:

```
0: 63
1: undefined
2: 16
3: undefined
4: undefined
```

Data Querying Methods

This topic describes data querying methods and objects that contain information. It includes the following topics:

Is Defined Method

The Is Defined method tests if a variable or object property is defined. It returns one of the following values:

- **The item is defined.** It returns the following value:

  True

- **The item is not defined.** It returns the following value:

  False

This method is unique to Siebel eScript. For more information, see "Make Sure the JavaScript Interpreter Can Run a Function" on page 57.

Format

`defined(`*var*`)`

Table 101 describes the arguments for the Is Defined method.

Table 101. Arguments for the Is Defined Method

Argument	Description
var	The variable or object property you must query.

Example

The following example includes two uses of the Is Defined method. The first use examines a variable named t. The second use examines an object named t.t:

```
var t = 1;
   if (defined(t))
      TheApplication().Trace("t is defined");
   else
      TheApplication().Trace("t is not defined");

   if (!defined(t.t))
      TheApplication().Trace("t.t is not defined"):
   else
      TheApplication().Trace("t.t is defined");
```

Related Topics

For more information, see "Undefine Method" on page 154.

Is Finite Method

The Is Finite method determines if the value that the value argument contains is a finite number. It returns one of the following values:

- **It can convert the value to a number.** It returns the following value:

  True

- **The value evaluates to any of the following items.** It returns False:
  - NaN
  - POSITIVE_INFINITY
  - NEGATIVE_INFINITY

For more information, see "NaN Numbers" on page 28.

This method uses the same argument as the Is NaN method. For more information, see Table 102 on page 176.

Format

`isFinite(`*value*`)`

Is NaN Method

The Is NaN method determines if the value that the value argument contains is a number. It returns one of the following values:

- **The value is a number.** It returns the following value:

  True

- **The value is not a number.** It returns the following value:

  False

If the value argument references an object, then the Is NaN method always returns true because an object reference is not a number. For more information on the number object, see "NaN Numbers" on page 28.

Format

`isNaN(`*`value`*`)`

Table 102 describes the arguments for the Is NaN method.

Table 102. Arguments for the Is NaN Method

Argument	Description
value	The variable or expression that this method evaluates.

Example

The following examples use the Is NaN method:

```
IsNaN("123abc") //returns true

IsNaN("123") //returns false

IsNaN("999888777123") //returns false

IsNaN("The answer is 42") //returns true
```

Related Topics

For more information, see "Is Finite Method" on page 175.

Exception Object

If an operation fails, then the Siebel eScript engine creates an exception in the exception object.

Table 103 describes the arguments for the exception object.

Table 103. Arguments for Exception Objects

Argument	Description
errCode	Contains the error number.
errText	Contains a textual description of the error.

Example

The following example includes an exception object:

```
try
}
   var oBO = TheApplication().GetService("Incorrect name");
}
catch (e)
}
   var sText = e.errText;
   var nCode = e.errCode;
}
```

Function Object

A Function object contains the definition of a function that you define in Siebel eScript. It returns the code that you configure this function to return. For more information, see "Return Statement of a Function Object" on page 178.

Format A

```
function funcName( [arg1 [, ..., argn]] )
{
     body
}
```

In format A you declare a function, and then call it in your code. It is the standard way to define a function.

Format B

```
var funcName = new Function([arg1 [, ..., argn,]] body );
```

In format B you explicitly create a function. If you use format B to create a function object, then Siebel CRM evaluates it each time it uses this function. This configuration is not as efficient as format A because Siebel CRM compiles a declared function only one time instead of evaluating it every time it uses the function.

Arguments

Table 104 describes the arguments for a function object.

Table 104. Arguments of a Function

Argument	Description
funcName	The name of the function.
arg1 [, …, argn]	An optional list of arguments that the function accepts.
body	The lines of code that the function runs.

Example 1

The following example uses format A to declare a function named AddTwoNumbers. It uses AddTwoNumbers as the name of the function:

```
function AddTwoNumbers (a, b)
{
   return (a + b);
}
```

Example 2

The following example uses format B to create a function named AddTwoNumbers. It uses the Function constructor to create a variable named AddTwoNumbers. The value of this variable is a reference to the function that the Function constructor creates:

```
AddTwoNumbers = new Function ("a", "b", "return (a + b)");
```

Length Property of a Function Object

The length property returns the number of arguments that the function expects.

Format

funcName.length

Table 105 describes the arguments for the length property.

Table 105. Arguments for the Length Property

Argument	Description
funcName	The name of the function that the length property uses to return the number of arguments.

Return Statement of a Function Object

The Return statement passes a value back to the function that called it.

Format

return *value*

Table 106 describes the arguments for the Return statement.

Table 106. Arguments for the Return Statement

Argument	Description
value	Contains a value from the function that calls the Return statement.

Usage

Siebel CRM does not run any code in a function that occurs after a Return statement.

If you define a return type for a custom function, then you must explicitly return a value of the same type that the function header specifies. All control paths must lead to a Return statement.

Example 1

The function in the following example returns a value that is equal to the number that Siebel CRM passes to it multiplied by 2, and then divided by 5:

```
function DoubleAndDivideBy5(a)
{
   return (a*2)/5
}
```

Example 2

The following example does the following work:

- Uses the value from the function in "Example 1" on page 179
- Calculates the following expression:

  ```
  n = (10 * 2) / 5 + (20 * 2) / 5
  ```

- Displays the value for n, which is 12:

  ```
  function myFunction()
  {
     var a = DoubleAndDivideBy5(10);
     var b = DoubleAndDivideBy5(20);
     TheApplication().RaiseErrorText(a + b);
  }
  ```

Mathematical Methods

This topic describes mathematical methods. It includes the following topics:

For more information on the number object, see "NaN Numbers" on page 28.

Overview of Mathematical Methods

Some math methods return data in radians. To convert radians to degrees, you can use the following formula:

radians multiplied by (180/Math.PI).

Properties of the Math Object

This topic describes properties of the math object.

Base E Property

The Base E property stores the number value for *e*, which is the base for natural logarithms. The value of *e* internally is approximately 2.7182818284590452354.

Format

```
Math.E
```

Logarithm 2 E Property

The Logarithm 2 E property stores the number value for the base 2 logarithm of *e*, which is the base of the natural logarithms. The value of the base 2 logarithm of *e* internally is approximately 1.4426950408889634. The value of the Logarithm 2 E property is approximately the reciprocal of the value of Math Logarithm 2 property.

Format

```
Math.LOG2E
```

Logarithm 10 E Property

The Logarithm 10 E property is the number value for the base 10 logarithm of *e*, which is the base of the natural logarithms. The value of the base 10 logarithm of *e* internally is approximately 0.4342944819032518. The value of the Logarithm 10 E property is approximately the reciprocal of the value of the Natural Logarithm 10 property.

Format

```
Math.LOG10E
```

Natural Logarithm 2 Property

The Natural Logarithm 2 property stores the number value for the natural logarithm of 2. The value of the natural logarithm of 2 internally is approximately 0.6931471805599453.

Format

```
Math.LN2
```

Math Natural Logarithm 10 Property

The Natural Logarithm 10 property stores the number value for the natural logarithm of 10. The value of the natural logarithm of 10 internally is approximately 2.302585092994046.

Format

```
Math.LN10
```

PI Property

The Pi property holds the number value for pi, which is the ratio of the circumference of a circle to the diameter of the circle. This value internally is approximately 3.14159265358979323846.

Format

```
Math.PI
```

Square Root 1/2 Property

The Square Root 1/2 property stores the number value for the square root of ½. This value internally is approximately 0.7071067811865476. The value of the Square Root 1/2 property is approximately the reciprocal of the value of the Square Root 2 property.

Format

```
Math.SQRT1_2
```

Square Root 2 Property

The Square Root 2 property stores the number value for the square root of 2. This value internally is approximately 1.4142135623730951.

Format

```
Math.SQRT2
```

Get Absolute Value Method

The Get Absolute Value method returns the absolute value of the value that the number argument contains. If it cannot convert this value to a number, then it returns NaN.

Format

```
Math.abs(number)
```

Table 107 describes the arguments for the Get Absolute Value method.

Table 107. Arguments for the Get Absolute Value Method

Argument	Description
number	A numeric literal or numeric variable.

Get Arc Cosine Method

The Get Arc Cosine method returns the arc cosine of the value that the number argument contains, expressed in radians from 0 to pi. If any of the following situations are true, then it returns NaN:

- The method cannot convert the value to a number.
- The value is greater than 1 or less than negative 1.

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

```
Math.acos(number)
```

Get Arcsine Method

The Get Arcsine method returns an approximate arcsine of the value that the number argument contains expressed in radians in the range of negative pi/2 through pi/2. If any of the following situations are true, then this method returns NaN:

- It cannot convert the value to a number.
- The value is greater than 1 or less than negative 1.

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

```
Math.asin(number)
```

Get Arctangent Method

The Get Arctangent method returns an approximate arctangent of the value that the number argument contains, expressed in radians and ranging from negative pi/2 through pi/2.

This method assumes the value that the number argument contains is the ratio of the following sides of a right triangle:

- The side that is opposite of the angle that this method must calculate
- The side that is adjacent to the angle

It returns a value for this ratio.

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

```
Math.atan(number)
```

Example for the Get Arctangent Method

The following example calculates the roof angles that are necessary for a house that includes the following dimensions:

- An attic ceiling height of 8 feet at the roof peak

- A 16 foot span from the outside wall to the center of the house

The Get Arctangent method returns the angle in radians. To convert the value to degrees, it multiplies it by 180/PI. To examine how the Get Arctangent method is different from the Get Arctangent 2 method, you can compare it to the example in the "Example for the Get Arctangent 2 Method" on page 184 topic. These examples return the same value:

```
function RoofBtn_Click ()
{
   var height = 8;
   var span = 16;
   var angle = Math.atan(height/span)*(180/Math.PI);

   TheApplication().RaiseErrorText("The angle is " +
      Clib.rsprintf("%5.2f", angle) + " degrees.")
}
```

Get Arctangent 2 Method

The Get Arctangent 2 method returns an approximate arctangent of the value that the y argument contains divided by the value that the x argument contains, expressed in radians and ranging from negative pi through pi.

To determine the quadrant of the result, this method uses the signs of the arguments. It is intentional and traditional that the argument named y is the first argument and the argument named x is the second argument.

Format

`Math.atan2(`*y, x*`)`

Table 108 describes the arguments for the Get Arctangent 2 method.

Table 108. Arguments for the Get Arctangent 2 Method

Argument	Description
y	The value on the y axis.
x	The value on the x axis.

Example for the Get Arctangent 2 Method

The following example finds the roof angle necessary for a house. It is identical to the example for the Get Arctangent method except this example uses the Get Arctangent 2 method. For more information, see "Example for the Get Arctangent Method" on page 183:

```
function RoofBtn2_Click ()
{
   var height = 8;
   var span = 16;
   var angle = Math.atan2(span, height)*(180/Math.PI);
```

```
      TheApplication().RaiseErrorText("The angle is " +
      Clib.rsprintf("%5.2f", angle) + " degrees.")
   }
```

Get Ceiling Method

The Get Ceiling method returns the smallest integer that is not less than the value that the number argument contains. If this argument already contains an integer, then this method returns the value of this argument. If it cannot convert the value to a number, then it returns the following value:

NaN

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

`Math.ceil(`*`number`*`)`

Example

The following example creates a random number between 0 and 100 and displays the integer range where the number falls. Each run of this code produces a different result:

```
var x = Math.random() * 100;
   TheApplication().RaiseErrorText("The number is between " +
      Math.floor(x) + " and " + Math.ceil(x) + ".");
```

Get Cosine Method

The Get Cosine method returns an approximate cosine of the value that the number argument contains, expressed in radians.The return value is between negative 1 and 1. The angle can be positive or negative. If this method cannot convert the value to a number, then it returns the following value:

NaN

This method uses the same argument as the Get Absolute Value method. The only difference is that the number argument for the Get Cosine method includes an angle in radians. For more information, see Table 107 on page 182.

Format

`Math.cos(`*`number`*`)`

Example

The following example finds the length of a roof, given the roof pitch and the distance of the house from the center of the house to the outside wall of the house:

```
function RoofBtn3_Click ()
{
   var pitch;
   var width;
   var roof;

   pitch = 35;
   pitch = Math.cos(pitch*(Math.PI/180));
   width = 75;
   width = width / 2;
   roof = width/pitch;

   TheApplication().RaiseErrorText("The length of the roof is " +
         Clib.rsprintf("%5.2f", roof) + " feet.");
}
```

Get Exponential Method

The Get Exponential method returns *e* raised to the power of x where:

- *e* is the base of the natural logarithms. The value of *e* internally is approximately 2.7182818284590452354.
- *x* is the value that the number argument contains.

If this method cannot convert the value that the number argument contains to a number, then it returns the following value:

NaN

Format

`Math.exp(`*`number`*`)`

Table 109 describes the arguments for the Get Exponential method.

Table 109. Arguments of the Get Exponential Method

Argument	Description
number	The exponent value of the base of e.

Related Topics

For more information, see the following topics:

- "Base E Property" on page 180
- "Math Natural Logarithm 10 Property" on page 181
- "Logarithm 10 E Property" on page 181
- "Get Logarithm Method" on page 187

- "Logarithm 2 E Property" on page 181
- "Logarithm 10 E Property" on page 181

Get Floor Method

The Get Floor method returns the greatest integer that is not greater than the value that the number argument contains. If this value is already an integer, then it returns the value that the number argument contains. If this method cannot convert the value that the number argument contains to a number, then it returns the following value:

NaN

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

`Math.floor(`*`number`*`)`

Example

For an example, see "Get Ceiling Method" on page 185.

Get Logarithm Method

The Get Logarithm method returns an approximate natural logarithm of the value that the number argument contains.

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

`Math.log(`*`number`*`)`

Using the Get Logarithm Method and Raise Power Method with Large Numbers

For a large number, you must use the Get Logarithm method. The number 999^1000 (999 to the 1000th power) is an example of a large number. If you use the Raise Power method instead of the Get Logarithm method with a large number, then the Raise Power method returns the following value:

```
Infinity
```

Example

This example uses the Get Logarithm method to determine which of the following numbers is larger:

- 999^1000 (999 to the 1000th power)

- 1000^999 (1000 to the 999th power):

```
function Test_Click ()
{
   var x = 999;
   var y = 1000;
   var a = y*(Math.log(x));
   var b = x*(Math.log(y))
   if ( a > b )
      TheApplication().
         RaiseErrorText("999^1000 is greater than 1000^999.");
   else
      TheApplication().
         RaiseErrorText("999^1000 is not greater than 1000^999.");
}
```

Get Maximum Method

The Get Maximum method returns the larger of the values in the x argument and the y argument. If it cannot convert the value that the number argument contains to a number, then it returns the following value:

NaN

Format

`Math.max(`*`x, y`*`)`

Table 110 describes the arguments for the Get Maximum method.

Table 110. Arguments for the Get Maximum Method

Argument	Description
x	A numeric literal or numeric variable.
y	A numeric literal or numeric variable.

Related Topics

For more information, see "Get Minimum Method" on page 188.

Get Minimum Method

The Get Minimum method returns the smaller of the values that the x argument and the y argument contain. If it cannot convert the value that the number argument contains to a number, then it returns the following value:

NaN

This method uses the same argument as the Math Maximum method. For more information, see Table 110 on page 188.

Format

```
Math.min(x, y)
```

Get Quotient Method

The Get Quotient method returns the quotient after a division operation that the Clib Divide method performs. You use this method in conjunction with the Clib Divide method.

Format

```
intVar.quot
```

Table 111 describes the arguments for the Get Quotient method.

Table 111. Arguments for the Get Quotient Method

Argument	Description
intVar	Any variable that contains an integer.

Example

For an example, see "Clib Divide Method" on page 266.

Get Random Number Method

The Get Random Number method creates, and then returns a pseudo-random number between 0 and 1. It uses no arguments.

Where possible, you must use the Get Random Number method instead of the Clib Create Random Number method. You use the Clib Create Random Number method only if you must use the Clib Initialize Random Number Generator method to create an initial value for the random number generator. For more information, see the following topics:

- "Clib Create Random Number Method" on page 265
- "Clib Initialize Random Number Generator Method" on page 270

Format

```
Math.random()
```

Example

The following example creates a random string of characters in a range. The Get Random Number method sets the range between lowercase letter a through lowercase letter z:

```
function Test_Click ()
{
   var str1 = "";
   var letter;
   var randomvalue;
   var upper = "z";
   var lower = "a";

   upper = upper.charCodeAt(0);
   lower = lower.charCodeAt(0);

   for (var x = 1; x < 26; x++)
   {
      randomvalue = Math.round(((upper - (lower + 1)) *
         Math.random())  + lower);
      letter = String.fromCharCode(randomvalue);
      str1 = str1 + letter;
   }

   TheApplication().RaiseErrorText(str1);
}
```

Get Remainder Method

The Get Remainder method returns the remainder after a division operation that the Clib Divide method performs. You use this method in conjunction with the Clib Divide method.

Format

intVar.rem

Table 112 describes the arguments for the Get Remainder method.

Table 112. Arguments for the Get Remainder Method

Argument	Description
intVar	Any variable that contains an integer.

Example

For an example, see “Clib Divide Method” on page 266.

Get Sine Method

The Get Sine method returns the sine of an angle, expressed in radians. It returns the sine of the value that the number argument contains. The return value is between negative 1 and 1. If this method cannot convert the value that the number argument contains, then it returns the following value:

NaN

Format

```
Math.sin(number)
```

Table 113 describes the arguments for the math sine method.

Table 113. Arguments for the Get Sine Method

Argument	Description
number	A numeric expression that contains a number that includes the size of an angle, expressed in radians. This number can be positive or negative.

Get Square Root Method

The Get Square Root method returns the square root of the value that the number argument contains. If the value that the number argument contains is a negative number or if this method cannot convert this value to a number, then it returns the following value:

NaN

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

```
Math.sqrt()
```

Get Tangent Method

The Get Tangent method returns the tangent of the value that the number argument contains. If it cannot convert the value that the number argument contains, then it returns the following value:

NaN

Format

```
Math.tan(number)
```

Table 114 describes the arguments for the Get Tangent method.

Table 114. Arguments for the Get Tangent Method

Argument	Description
number	A numeric expression that contains the number of radians in the angle whose tangent this method returns.

Raise Power Method

The Raise Power method raises the value that the x argument contains to the power of the value that the y argument contains. It returns the result in the x argument. For more information, see "Using the Get Logarithm Method and Raise Power Method with Large Numbers" on page 187.

Format

`Math.pow(`*x, y*`)`

Table 115 describes the arguments for the Raise Power method.

Table 115. Arguments of the Raise Power Method

Argument	Description
x	The number that this method raises.
y	The power to which this method raises the value that the x argument contains.

Example

This example uses the Raise Power method to determine which of the following numbers is larger:

- 99^100 (99 to the 100th power)
- 100^99 (100 to the 99th power):

```
function Test_Click ()
{
   var a = Math.pow(99, 100);
   var b = Math.pow(100, 99);
   if ( a > b )
      TheApplication().RaiseErrorText("99^100 is greater than 100^99.");
   else
      TheApplication().RaiseErrorText("99^100 is not greater than 100^99.");
}
```

Round Number Method

The Round Number method does the following:

- If the fractional part is equal to or greater than 0.5, then it rounds the value in the number argument up.
- If the fractional part is less than 0.5, then it rounds the value in the number argument down.

It rounds a positive number or a negative number to the nearest integer.

It returns the integer that is closest in value to the value that the number argument contains.

This method uses the same argument as the Get Absolute Value method. For more information, see Table 107 on page 182.

Format

```
Math.round(number)
```

Example

The following example uses the Round Number method:

```
var a = Math.round(123.6);
var b = Math.round(-123.6)
TheApplication().RaiseErrorText(a + "\n" + b)
```

This example provides the following results:

```
124
negative 124
```

Avoiding Precision Loss Due to Rounding

The following example illustrates precision loss due to rounding:

```
var n = 34.855;
n = n* 100;
var r = Math.round(n)
```

The value of the n variable is 3485.499999999999995 instead of 3485.5. Rounding this value results in a value of 3485 instead of 3486.

The following example avoids loss of precision due to rounding:

```
var n = parseFloat(34.855);
n = parseFloat(n1b*100.0);
var r = Math.round(n);
```

If you multiply or divide a value, and then round that value, then rounding might not be precise. Multiplication and division can cause precision loss.

Regular Expression Methods

This topic describes regular expression methods. It includes the following topics:

Overview of Regular Expression Methods

A *regular expression* is an object instance of a character pattern that is associated with attributes that ECMAScript uses to perform a character pattern search of a string. A regular expression uses the following short format:

```
RegExp
```

For more information, see ECMAScript specifications.

The Siebel T eScript engine supports the following methods of the regular expression object. The Siebel ST eScript engine does not support these methods:

- RegExp.$n, including '$_' and '$&'
- RegExp.input
- RegExp.lastMatch
- RegExp.lastParen
- RegExp.leftContext
- RegExp.rightContext

If you must use ST eScript code, then instead of using one of these methods you must modify your script to use an equivalent function on the target object.

Properties of Regular Expressions

This topic describes properties of regular expressions. The Siebel ST eScript engine and the Siebel T eScript engine supports these properties. Throughout this topic, the term *regexp* represents an object instance of a regular expression.

You can write code that uses the Compile Regular Expressions method to modify the attribute of a regular expression instance for one of these properties. For example, if you must write code that modifies the global attribute of a regular expression instance. For more information, see "Compile Regular Expressions Method" on page 196.

Regular Expression Global Property

The Regular Expression Global property is a read-only property that indicates the value of the global attribute of an instance of the regular expression object. The value it returns depends on the attribute:

- **The value g is an attribute of the regular expression.** It returns the following value:

  True

- **The value g is not an attribute of the regular expression.** It returns the following value:

  False

Format

regexp.global

Example

The following example uses the regular expression global property:

```
// Create RegExp instance with global attribute.
var pat = /^Begin/g;
//or
var pat = new RegExp("^Begin", "g");
//Then pat.global == true.
```

Regular Expression Ignore Case Property

The Regular Expression Ignore Case property is a read-only property that indicates the value of the ignoreCase attribute of an instance of the regular expression object. The value it returns depends on the attribute:

- **The value i is an attribute of the regular expression.** It returns the following value:

  True

- **The value i is not an attribute of the regular expression.** It returns the following value:

  False

Format

regexp.ignoreCase

Example

The following example uses the Regular Expression Ignore Case property:

```
// Create RegExp instance with ignoreCase attribute.
var pat = /^Begin/i;
//or
var pat = new RegExp("^Begin", "i");
//Then pat.ignoreCase == true.
```

Regular Expression Multiline Property

The Regular Expression Multiline property is a read-only property that indicates the value of the multiline attribute of an instance of the regular expression object. It determines if Siebel CRM performs a pattern search in multiline mode. The value it returns depends on the attribute:

- **The value m is an attribute of the regular expression.** It returns the following value:

  True

- **The value m is not an attribute of the regular expression.** It returns the following value:

  False

Format

regexp.multiline

Example

The following example uses the Regular Expression Multiline property:

```
// Create RegExp instance with multiline attribute.
var pat = /^Begin/m;
//or
var pat = new RegExp("^Begin", "m");
//Then pat.multiline == true.
```

Regular Expression Source Property

The Regular Expression Source property is a read-only property that stores the regular expression that Siebel CRM uses to find matches in a string, not including the attributes.

Format

regexp.source

Example

The following example uses the Regular Expression Source property:

```
var pat = /t.o/g;
// Then pat.source == "t.o"
```

Compile Regular Expressions Method

The Compile Regular Expressions method modifies the pattern and attributes for the current instance of a regular expression object. It allows you to use a regular expression instance multiple times with modifications to the characteristics of this instance. You use it with a regular expression that the constructor function creates. You do not use this method with the literal notation.

Format

regexp.compile(*pattern*[, *attributes*])

Table 116 describes the arguments for the Compile Regular Expressions method.

Table 116. Arguments of the Compile Regular Expressions Method

Argument	Description
pattern	A string that contains a new regular expression.
attributes	A string that contains new attributes. If you include the attributes argument, then this string must be empty, or it must contain one or more of the following characters: ■ **i.** Sets the ignoreCase property to true. ■ **g.** Sets the global property to true. ■ **m.** Sets the multiline property to true.

Example

The following example uses the Compile Regular Expressions method:

```
var regobj = new RegExp("now");
// use this RegExp object
regobj.compile("r*t");
// use it some more
regobj.compile("t.+o", "ig");
// use it some more
```

Related Topics

For more information, see the following topics:

- “Regular Expression Global Property” on page 194
- “Regular Expression Ignore Case Property” on page 195
- “Regular Expression Multiline Property” on page 196
- “Regular Expression Source Property” on page 196

Get Regular Expression from String Method

The Get Regular Expression from String method searches the string that you specify in the str argument for a regular expression. It returns one of the following depending on if it finds this regular expression:

- **It finds the regular expression.** It returns an array of strings that includes information about each match it finds and the property sets for these matches.

- **It does not find the regular expression.** It returns the following value:

  Null

Format

regexp.exec(*str*)

Table 117 describes the arguments for the Get Regular Expression from String method.

Table 117. Arguments for the Get Regular Expression from String Method

Argument	Description
str	A string that this method searches for a regular expression.

Usage Without Setting the Global Attribute

Assume you configure Siebel CRM to run the Get Regular Expression from String method, you do not set the g global attribute on the regular expression instance, and the method finds a match. In this situation, the array elements that it returns include the following information:

- **Element 0.** The first text in the string that matches the primary regular expression.
- **Element 1.** The text that the first subpattern of the regular expression instance matches. It encloses this subpattern in parentheses.
- **Element 2 through element n.** Each subsequent element uses the same format as element 1.

The returned array includes the following properties:

- **Length property.** The number of text matches that exist in the returned array.
- **Index property.** The start position of the first text that matches the primary regular expression.
- **Input property.** The target string that the method searched.

Usage With Setting the Global Attribute

Assume you configure Siebel CRM to run the Get Regular Expression from String method but you do set the g global attribute on the regular expression instance. In this situation, this method returns the same result as if the global attribute is not set but the behavior is more complex, which allows more operations. It does the following work:

1. Begins searching at the position in the target string that the this.lastIndex property specifies.
2. After it finds a match, it sets the this.lastIndex property to the position after the last character in the matched text.

The this.lastIndex property possesses read and write capabilities. To find all matches of a pattern, you can configure this method to set the this.lastIndex property to the start position of the previous match that it found plus 1. This configuration causes this method to loop through a string. When it does not find a match, it resets the this.lastIndex property to 0.

Using the Get Regular Expression from String Method with the T eScript Engine

If you use code that you create with the T eScript engine, and if the Get Regular Expression from String method finds a match, then it sets the appropriate static properties of the regular expression object. For example, it sets the following properties:

- RegExp.leftContext
- RegExp.rightContext
- RegExp.$n
- And so on

This configuration provides more information about the matches.

Using the Get Regular Expression from String Method and the Get Regular Expression from StringVar Method

The behavior of the Get Regular Expression from String method and the Get Regular Expression from StringVar method varies depending on if you set the global attribute on the regular expression:

- **You do not set the global attribute.** The methods return the same array. The return values and the index and input properties are the same.
- **You do set the global attribute.** The methods return different arrays.

For more information, see "Get Character From String Method" on page 91.

Example 1

The following example calls the Get Regular Expression from String method from a regular expression whose global attribute is not set:

```
function fn ()
{
   var myString = new String("Better internet");
   var myRE = new RegExp(/(.).(.er)/i);
   var results = myRE.exec(myString);
   var resultmsg = "";
   for(var i =0; i < results.length; i++)
   {
      resultmsg = resultmsg + "return[" + i + "] = " + results[i] + "\n";
   }
   TheApplication().RaiseErrorText(resultmsg);
}
fn ();
```

This example provides the following output:

```
return[0] = etter   \\First text that contains primary pattern ...er (any three
                    \\characters followed by "er")
return[1] = e        \\First text matching the first subpattern (.) (any single
                    \\character) in the first text matching the primary pattern
return[2] = ter      \\First text matching the second subpattern (.er) (any single
                     \\character followed by "er") in the first text matching
                     \\the primary pattern
```

Example 2

The following example calls the Get Regular Expression from String method from a regular expression whose global attribute is set. This method returns all matches that exist of the primary pattern in a string of the regular expression, including matches that overlap:

```
function fn ()
{
   var str = "ttttot tto";
   var pat = new RegExp("t.t", "g");
   var resultmsg = "";
   while ((rtn = pat.exec(str)) != null)
   {
      resultmsg = resultmsg + "Text = " + rtn[0] + " Pos = " + rtn.index
      + " End = " + (pat.lastIndex - 1) + "\n";
      pat.lastIndex = rtn.index + 1;
   }
   TheApplication().RaiseErrorText(resultmsg)
}
fn ();
```

This example provides the following output:

```
Text = ttt Pos = 0 End = 2
Text = ttt Pos = 1 End = 3
Text = tot Pos = 3 End = 5
Text = t t Pos = 5 End = 7
```

Related Topics

For more information, see the following topics:

- "Get Character From String Method" on page 91
- "Is Regular Expression in String Method" on page 200

Is Regular Expression in String Method

The Is Regular Expression in String method determines if a string includes a regular expression. It returns one of the following values:

- If the string includes a regular expression, then it returns the following value:

  True

- If the string does not include a regular expression, then it returns the following value:

  False

This method uses the same arguments as the Get Regular Expression from String method. For more information, see Table 117 on page 198.

Format

regexp.test(*str*)

Usage

The Is Regular Expression in String method is equivalent to *regexp*.exec(*str*)!=null.

Usage for this method with T eScript code is the same as usage for the Get Regular Expression from String method. For more information, see "Using the Get Regular Expression from String Method with the T eScript Engine" on page 199.

You can write code that uses the Is Regular Expression in String method with the g global attribute set on the regular expression instance. This functionality uses the lastIndex property in the same way as the Get Regular Expression from String method. For more information, see "Usage With Setting the Global Attribute" on page 198.

Example

The following example includes the Is Regular Expression in String method:

```
var str = "one two three tio one";
var pat = /t.o/;
rtn = pat.test(str);
// Then rtn == true.
```

Siebel Library Methods

This topic describes the Siebel library methods that Siebel eScript uses to call external libraries and applications. It includes the following topics:

Siebel Library Call DLL Method

The Siebel Library Call DLL method calls a procedure from a dynamic link library in Microsoft Windows or a shared object in UNIX. It returns an integer.

Windows Format

SElib.dynamicLink(*Library, Procedure, Convention*[*, [desc,] arg1, arg2, arg3, ..., arg*n])

UNIX Format

SElib.dynamicLink(*Library, Procedure*[*, arg1, arg2, arg3, ...arg*n])

In UNIX, Siebel CRM cannot use the Siebel Library Call DLL method to pass more than 22 arguments. These 22 arguments include the shared library name and the procedure name. You can configure Siebel CRM to pass up to 20 more arguments.

Table 118 describes the arguments for the Siebel Library Call DLL method.

Table 118. Arguments for the Siebel Library Call DLL Method

Argument	Description
Library	The library argument can include the following: ■ In Microsoft Windows, the name of the DLL that contains the procedure. ■ In UNIX, the name of a shared object. You must specify the fully qualified path name.
Procedure	The name or ordinal number of the procedure in the library dynamic link method.
Convention	The calling convention.
desc	Passes a Unicode string. For example, WCHAR.
arg1, arg2, arg3, ..., argn	Arguments for the Siebel Library Call DLL method.

Usage for the Convention Argument

Table 119 describes the calling conventions you must use with the Siebel Library Call DLL method.

Table 119. Calling Conventions for the Siebel Library Call DLL Method

Value	Description
CDECL STDCALL	Send the argument that appears last in the list first. For example, consider the following format: SElib.dynamicLink(*Library, Procedure, Convention*[*, [desc,] arg1, arg2, arg3*]) If arg1, arg2, and arg3 are defined, then this method sends the arguments in the following order: ■ arg3 ■ arg2 ■ arg1 The caller reads the arguments. The STDCALL value is almost always used in Win32.
PASCAL	Send the argument that appears first in the list first. For example, consider the following format: SElib.dynamicLink(*Library, Procedure, Convention*[*, [desc,] arg1, arg2, arg3*]) If arg1, arg2, and arg3 are defined, then this method sends the arguments in the following order: ■ arg1 ■ arg2 ■ arg3 The callee reads the arguments.

Usage if An Argument Is Not Defined

Siebel CRM passes values as 32-bit values. If an argument is not defined when Siebel CRM calls the Siebel Library Call DLL method, then it assumes that the argument is a 32-bit value. It passes the address of a 32-bit data element to the Siebel Library Call DLL method. This method then sets the value.

Usage If an Argument Is a Structure

SELib is a feature that Siebel eScript uses to call functions in the native DLLs. These DLLs can contain functions implemented in a third party language, such as C or C++. In this situation, an argument can include a structure.

If an argument is a structure, then it must include a structure that defines the binary data types in memory. Siebel CRM does the following:

1. Copies the structure to a binary buffer.
2. Calls the method.
3. Converts the binary data back into the data structure according to the rules defined in the Write BLOB Data method and the Clib Read From File method. It performs data conversion according to the current BigEndianMode setting.

   For more information, see "Write BLOB Data Method" on page 106 and "Clib Read From File Method" on page 240.

Example 1

The following example describes a proxy DLL that uses denormalized input values, creates the structure, and calls a method in the destination DLL:

```
#include <windows.h>
_declspec(dllexport)  int __cdecl
score (
   double AGE,
   double AVGCHECKBALANCE,
   double AVGSAVINGSBALANCE,
   double CHURN_SCORE,
   double CONTACT_LENGTH,
   double HOMEOWNER,
   double *P_CHURN_SCORE,
   double *R_CHURN_SCORE,
   char   _WARN_[5] )
{
   *P_CHURN_SCORE = AGE + AVGCHECKBALANCE + AVGSAVINGSBALANCE;
   *R_CHURN_SCORE =  CHURN_SCORE + CONTACT_LENGTH + HOMEOWNER;
   strcpy(_WARN_, "SFD");
   return(1);
}
```

Example 2

The following example calls a DLL. This code uses the buffer for pointers and characters:

```
function TestDLLCall3()
{
   var AGE = 10;
   var AVGCHECKBALANCE = 20;
   var AVGSAVINGSBALANCE = 30;
   var CHURN_SCORE = 40;
   var CONTACT_LENGTH = 50;
   var HOMEOWNER = 60;
   var P_CHURN_SCORE = Buffer(8);
   var R_CHURN_SCORE = Buffer(8);
   var _WARN_ = Buffer(5);
```

```
SElib.dynamicLink("jddll.dll", "score", CDECL,
   FLOAT64, AGE,
   FLOAT64, AVGCHECKBALANCE,
   FLOAT64, AVGSAVINGSBALANCE,
   FLOAT64, CHURN_SCORE,
   FLOAT64, CONTACT_LENGTH,
   FLOAT64, HOMEOWNER,
   P_CHURN_SCORE,
   R_CHURN_SCORE,
   _WARN_);

    var r_churn_score = R_CHURN_SCORE.getValue(8, "float");
    var p_churn_score = P_CHURN_SCORE.getValue(8, "float");
    var nReturns = r_churn_score + p_churn_score;
    return(nReturns);
    }
```

Other Examples

The following example calls a DLL function in the default codepage:

```
var sHello = "Hello";
Selib.dynamicLink("MyLib.dll", "MyFunc", CDECL,  sHello);
```

The following example calls a DLL function that passes Unicode strings:

```
var sHello = "Hello";
Selib.dynamicLink("MyLib.dll", "MyFunc", CDECL,  WCHAR, sHello);
```

The following example calls a DLL function that passes Unicode and nonUnicode strings:

```
var sHello = "Hello";
var sWorld = "world";
Selib.dynamicLink("MyLib.dll", "MyFunc", CDECL,  WCHAR, sHello, sWorld);
```

The following example calls an external application and passes arguments 0, 0, and 5 to it:

```
SElib.dynamicLink("shell32", "ShellExecuteA", STDCALL, 0, "open",
"c:\\Grabdata.exe", 0, 0, 5).
```

Related Topics

For more information, see “Clib Send Command Method” on page 291.

Siebel Library Get Pointer Address Method

The Siebel Library Get Pointer Address method gets the address in memory of the first byte of data in a buffer variable. It returns the address of the pointer to the buffer variable. For more information, see "Buffer Methods" on page 108.

CAUTION: A pointer is valid only until a script modifies the variable that the bufferVar argument identifies or until the variable goes out of scope in a script. Placing data in the memory that this variable occupies after such a modification is not recommended. Be careful not to place more data than this memory can hold.

Format

`SElib.pointer(`*`bufferVar`*`])`

Table 120 describes the arguments for the Siebel Library Get Pointer Address method.

Table 120. Arguments for the Siebel Library Get Pointer Address Method

Argument	Description
bufferVar	The name of a buffer variable.

Example

The following example includes the Siebel Library Get Pointer Address method:

```
TheApplication().TraceOn("c:\\eScript_trace.txt","allocation","all");
var v = new Buffer("Now");
// Collect "Now", the original value, for display.
TheApplication().Trace(v);
// Get the address of the first byte of v, "N"
var vPtr = SElib.pointer(v);
// Get the "N"
var p = SElib.peek(vPtr);
// Convert "N" to "P"
SElib.poke(vPtr,p+2);
// Display "Pow"
TheApplication().Trace(v);
TheApplication().TraceOff();
```

This example produces the following output:

```
COMMENT,Now
COMMENT,Pow
```

Related Topics

For more information, see the following topics:

- "BLOB Methods" on page 101
- "Clib Get Memory Method" on page 262

Siebel Library Peek Method

The Siebel Library Peek method reads, and then returns data from a position in memory.

Format

`SElib.peek(`*`address`*`[, `*`dataType`*`])`

Table 121 describes the arguments for the Siebel Library Peek method.

Table 121. Arguments for the Siebel Library Peek Method

Argument	Description
address	Identifies the address in memory that this method uses to read data.
dataType	The type of data that this method returns. You can specify one of the following types: ■ UWORD8 ■ SWORD8 ■ UWORD16 ■ SWORD16 ■ UWORD24 ■ SWORD24 ■ UWORD32 ■ SWORD32 ■ FLOAT32 ■ FLOAT64 ■ FLOAT80 The default value is UWORD8. You can add the following prefix on some types: ■ S for signed ■ U for unsigned The numeric suffix specifies the number of bytes to get. An example of a numeric suffix is 8 or 16. FLOAT80 is not available in Win32.

Example

The following example uses the Siebel Library Peek method:

```
TheApplication().TraceOn("c:\\eScript_trace.txt","allocation","all");
var v = new Buffer("Now");
// Collect "Now", the original value, for display.
TheApplication().Trace(v);
// Get the address of the first byte of v, "N"
var vPtr = SElib.pointer(v);
// Get the "N"
var p = SElib.peek(vPtr);
// Convert "N" to "P"
SElib.poke(vPtr,p+2);
// Display "Pow"
TheApplication().Trace(v);
TheApplication().TraceOff();
```

This example produces the following output:

```
COMMENT,Now
COMMENT,Pow
```

Related Topics

For more information, see the following topics:

- "Get BLOB Data Method" on page 103
- "Clib Get Memory Method" on page 262
- "Clib Read From File Method" on page 240

Siebel Library Write Data Method

The Siebel Library Write Data method writes data to a specific position in memory. It returns the address of the byte that immediately follows the data that it writes.

CAUTION: If your code directly accesses memory, then you must use this code with caution. To avoid moving data unexpectedly, you must clearly understand how the Siebel Library Write Data method affects memory.

Format

`SElib.poke(`*address*`, `*data*`[, `*dataType*`])`

Table 122 describes the arguments for the Siebel Library Write Data method.

Table 122. Arguments for the Siebel Library Write Data Method

Argument	Description
address	The starting address in memory where this method writes data.

Table 122. Arguments for the Siebel Library Write Data Method

Argument	Description
data	The data that this method writes in memory. The data type of this data must match the type that you specify in the dataType argument.
dataType	For more information, see Table 121 on page 207.

Example

The following example includes the Siebel Library Write Data method:

```
TheApplication().TraceOn("c:\\eScript_trace.txt","allocation","all");
var v = new Buffer("Now");
// Collect "Now", the original value, for display.
TheApplication().Trace(v);
// Get the address of the first byte of v, "N"
var vPtr = SElib.pointer(v);
// Get the "N"
var p = SElib.peek(vPtr);
// Convert "N" to "P"
SElib.poke(vPtr,p+2);
// Display "Pow"
TheApplication().Trace(v);
TheApplication().TraceOff();
```

This example produces the following output:

```
COMMENT,Now
COMMENT,Pow
```

Related Topics

For more information, see the following topics:

- "Write BLOB Data Method" on page 106
- "Clib Get Memory Method" on page 262
- "Clib Read From File Method" on page 240

Custom Methods

This topic describes custom methods. It includes the following topics:

Overview of Custom Methods

You can group variables and functions together in one variable, and then reference them as a group. A compound variable of this sort is an object where each individual item of the object is a property.

An object property is similar to a variable or a constant. An object method is similar to a function. To reference an object property, you use the name of the object and the name of the property, separated by a period:

object name.property

You can write code that uses any valid variable name as a property name. The following example assigns values to the width and height properties of a rectangle object, calculates the area of a rectangle, and then displays the result:

```
var Rectangle;

Rectangle.height = 4;
Rectangle.width = 6;

TheApplication().RaiseErrorText(Rectangle.height * Rectangle.width);
```

An object allows you to work with groups of data in a consistent way. For example, instead of using a single object named Rectangle, you can use multiple Rectangle objects, where each of these objects includes a separate value for width and a separate value for height.

How the Constructor Function Creates an Object

A constructor function creates an object template. To create a rectangle object, the following example uses a constructor function:

```
function Rectangle(width, height)
{
   this.width = width;
   this.height = height;
}
```

The following keyword references the arguments that the constructor function receives:

this

You can think of the this keyword as meaning *this object*.

Example of Using a Constructor Function

To create a rectangle object, the following example uses the new operator to call the constructor function:

```
var joe = new Rectangle(3,4)
var sally = new Rectangle(5,3);
```

This code creates the following rectangle objects:

- Joe, with a width of 3 and a height of 4
- Sally, with a width of 5 and a height of 3

This example creates a Rectangle class and two instances of this class. A constructor function creates objects that belong to the same class. Every object that a constructor function creates is an instance of that class.

Class instances share the same properties, although a single class instance can possess more unique properties. For example, adding the following code to the example adds a motto property to the joe rectangle. The sally rectangle does not include a motto property:

```
joe.motto = "Be prepared!";
```

How a Function Is Assigned to an Object

An object can contain a function and variables. A function assigned to an object is a method of that object.

A method uses the *this* operator to reference a method variable. The following example is a method that computes the area of a rectangle:

```
function rectangle_area()
{
   return this.width * this.height;
}
```

Siebel CRM passes no arguments to this function, so it is meaningless unless an object calls it. This object provides values for this.width and this.height.

The following code assigns a method to an object:

```
joe.area = rectangle_area;
```

The function now uses the values for height and width that were defined when you created the joe rectangle object.

To assign a method in a constructor function, you can use the *this* keyword. The following example creates an object class named Rectangle that includes the rectangle_area method as a property:

```
function rectangle_area()
{
   return this.width * this.height;
}

function Rectangle(width, height)
{
   this.width = width;
   this.height = height;
   this.area = rectangle_area;
}
```

The method is available to any instance of the class. The following example sets the value of area1 to 12 and the value of area2 to 15:

```
var joe = Rectangle(3,4);
var sally = Rectangle(5,3);

var area1 = joe.area();
var area2 = sally.area();
```

About Object Prototypes

An object prototype lets you specify a set of default values for an object. If Siebel eScript accesses an object property that is not assigned a value, then it consults the prototype. If this property exists in the prototype, and if this property contains a value, then Siebel eScript uses that value for the object property.

How an Object Prototype Conserves Memory

An object prototype helps you to make sure that every instance of an object uses the same default values and that these instances conserve the amount of memory that Siebel CRM requires to run a script. The joe and sally rectangles are each assigned an area method when they are created in "How a Function Is Assigned to an Object" on page 211. Siebel CRM allocates memory for this function twice, even though the method is exactly the same in each instance. To avoid this redundant memory, you can place the shared function or property in an object prototype. In this situation, every instance of the object uses the same function instead of each instance using a copy of the function.

Example of Using an Object Prototype

The following example creates a rectangle object with an area method in a prototype:

```
function rectangle_area()
{
   return this.width * this.height;
}

function Rectangle(width, height)
{
   this.width = width;
   this.height = height;
}

Rectangle.prototype.area = rectangle_area;
```

The following code can now reference the rectangle_area method as a method of any Rectangle object:

```
var area1 = joe.area();
var area2 = sally.area();
```

Adding Methods and Data to an Object Prototype

You can write code that adds methods and data to an object prototype at any time. You must define the object class but you do not have to create an instance of the object before you assign prototype values to it. If you assign a method or data to an object prototype, then Siebel CRM updates every instance of that object to include the prototype.

If you attempt to write to a property that Siebel CRM assigns through a prototype, then it creates a new variable for the newly assigned value. It uses this value for the value of this instance of the object property. Other instances of the object still refer to the prototype for their values. The following example specifies joe as a special rectangle whose area is equal to three times the width plus half the height:

```
function joe_area()
{
   return (this.width * 3) + (this.height/2);
}
joe.area = joe_area;
```

This code creates a value for joe.area that supersedes the prototype value. In this example, this value is a function. The sally.area property is still the default value that this prototype defines. The joe instance uses the new definition for the area method.

You cannot write code that declares a prototype in a function scope.

6 C Language Library Reference

This chapter describes reference information for the C language library you can use in Siebel eScript. It includes the following topics:

Overview of the Clib Object

The Clib (C library) object includes functions that are part of the standard library of the C programming language. It includes methods that can reference files, directories, strings, the environment, memory, and characters. It also includes time functions, error functions, sorting functions, and math functions. Siebel CRM supports the Clib library in Windows servers and Unix servers. It does not support the Clib library for Browser script.

The Clib object is a wrapper you can use to call a function in the standard C library as implemented for a specific operating system. The methods that this chapter describes might behave differently on different operating systems.

Using Siebel eScript Methods Instead of Clib Methods

Table 123 lists each Clib method that has an equivalent method in Siebel eScript. These methods are redundant because their functionality already exists in Siebel eScript. Where possible, you must use the Siebel eScript method instead of the equivalent Clib method. In some situations the Clib method is preferred and is more consistent in a section of script. For example, when working with a string routine that expects a null string.

Table 123. Siebel eScript You Can Use Instead of Clib Methods

Siebel eScript Method	Clib Method	Description
Get Absolute Value Method	abs	Calculates absolute value.
Get Arc Cosine Method	acos	Calculates the arc cosine.
Get Arcsine Method	asin	Calculates the arc sine.
Get Arctangent Method	atan	Calculates the arc tangent.
Get Arctangent 2 Method	atan2	Calculates the arc tangent of a fraction.
Convert String to Floating-Point Number Method	atof	Converts a string to a floating-point number.
Convert String to Integer Method	atoi	Converts a string to an integer.
Automatic conversion	atol	Converts a string to a long integer.
Get Ceiling Method	ceil	Rounds a number up to the nearest integer.
Get Cosine Method	cos	Calculates the cosine.
Get Exponential Method	exp	Calculates the exponential function.
Math absolute	fabs	Calculates the absolute value of a floating-point number.
Get Floor Method	floor	Rounds a number down to the nearest integer.
% operator, modulo	fmod	Calculates the remainder.
Math absolute	labs	Returns the absolute value of a long.
Get Logarithm Method	log	Calculates the natural logarithm.
Get Maximum Method	max	Returns the largest of one or more values.
Get Minimum Method	min	Returns the smallest of one or more values.
Raise Power Method	pow	Calculates x to the power of y.
Get Sine Method	sin	Calculates the sine.
Get Square Root Method	sqrt	Calculates the square root.
+ operator	strcat	Appends one string to another.
== operator	strcmp	Compares two strings.

Table 123. Siebel eScript You Can Use Instead of Clib Methods

Siebel eScript Method	Clib Method	Description
= operator	strcpy	Copies a string.
Get String Length Method	strlen	Gets the length of a string.
Change String to Lowercase Method	strlwr	Converts a string to lowercase.
Automatic conversion	strtod	Converts a string to decimal.
Automatic conversion	strtol	Converts a string to long.
Change String to Uppercase Method	strupr	Converts a string to uppercase.
Get Tangent Method	tan	Calculates the tangent.
string.toLowerCase	tolower	Converts a character to lowercase.
string.toUpperCase	toupper	Converts a character to uppercase.

The phrase *automatic conversion* in the Siebel eScript Method column means that Siebel eScript implicitly performs a conversion. For example, when comparing Siebel eScript to the atol Clib method, if the variable that will hold the converted string is of type Number, then Siebel eScript implicitly converts the string from a string to a long integer.

Clib File and Directory Methods

This topic describes Clib methods to create, open, lock, close, and delete files. It also describes methods to manipulate directories. It includes the following topics:

Overview of Clib File and Directory Methods

Siebel eScript can interpret a backslash (\) as a character combination. If you create a Windows path name, then you must include two backslashes to prevent this interpretation. For example:

- To change the working directory to C:\Applications\Myfolder, you use the following command:

```
Clib.chdir(“C:\\Applications\\Myfolder”);
```

- To use a UNC path to access a computer on your network, use four backslashes (\\\\) before the computer name:

```
Clib.system("copy \\\\server01\\share\\SR.txt D:\\SR.txt ");
```

For general usage information that applies to these methods, see “Overview of Clib File Input and Output Methods” on page 230.

Clib Close File Method

The Clib Close File method writes to disk the data that currently resides in the buffer for a file. It then closes this file. It returns one of the following values:

- If successful, then it returns the following value:

  Zero

- If not successful, then it returns the following value:

  EOF

The file pointer is no longer valid after this call.

Format

`Clib.fclose(`*`filePointer`*`)`

The arguments for this method are the same as the arguments for the Clib Clear Error method. For more information, see “Arguments for the Clib Clear Error Method” on page 286.

Example

The following example creates and writes to a text file, and then closes this file. It also tests for an error condition. If an error occurs, then it displays a message and clears the buffer:

```
function Test_Click ()
{
   var fp = Clib.fopen('c:\\temp000.txt', 'wt');
   Clib.fputs('abcdefg\nABCDEFG\n', fp);
   if (Clib.fclose(fp) != 0)
   {
      TheApplication().RaiseErrorText('Unable to close file.' +
         '\nContents are lost.');
   }
```

```
      else
         Clib.remove('c:\\temp000.txt');
   }
```

For more information, see "Clib Clear Buffer Method" on page 236.

Clib Create Temporary File Method

The Clib Create Temporary File method creates and opens a temporary binary file. It automatically removes the file pointer and the temporary file when Siebel CRM closes the file or when the code finishes. It returns one of the following values:

- If successful, then it returns the file pointer of the file that it created.
- If not successful, then it returns the following value:

  Null

The location of where it creates the temporary file depends on how Clib is implemented on the operating system you use.

Format

```
Clib.tmpfile()
```

Example

For an example, see "Clib Get Characters to Next Line Method" on page 237.

Clib Create Temporary File Name Method

The Clib Create Temporary File Name method creates a temporary file name. This name is not the same as the name of any existing file and it is not the same as any file name that this method returns while this code runs. It returns the file name as a string in the str argument.

Format

```
Clib.tmpnam([str])
```

Table 124 describes the arguments for the Clib Create Temporary File Name method.

Table 124. Arguments for the Clib Create Temporary File Name Method

Argument	Description
str	A container that holds the name of the temporary file.

Clib Delete File Method

The Clib Delete File method deletes a file. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns the following value:

  Negative 1

Format

`Clib.remove(`*`filename`*`)`

Table 125 describes the arguments for the Clib Delete File method.

Table 125. Arguments for the Clib Delete File Method

Argument	Description
filename	A string or string variable that contains the name of the file that this method deletes.

Clib Lock File Method

The Clib Lock File method locks or unlocks a file for simultaneous use by multiple processes. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns a nonzero integer.

Format

`Clib.flock(`*`filePointer, mode`*`)`

Table 126 describes the arguments for the Clib Lock File method.

Table 126. Arguments for the Clib Lock File Method

Argument	Description
filePointer	The file that this method locks or unlocks. The Clib Open File method or the Clib Create Temporary File method can return this file name.
mode	You can specify one of the following values: ■ **LOCK_EX.** Lock for exclusive use. ■ **LOCK_SH.** Lock for shared use. ■ **LOCK_UN.** Unlock. ■ **LOCK_NB.** Nonblock.

Usage

The Clib Lock File method applies or removes an advisory lock on the file that the filePointer argument identifies. An *advisory lock* is a type of lock that allows cooperating processes to perform consistent operations on a file. Other processes might still reference the files, so inconsistencies might occur.

Locking allows the following types of locks:

- **Shared lock.** Multiple processes can use shared locks on the same file at the same time. Read permission is required to obtain a shared lock.
- **Exclusive lock.** The following configurations cannot exist on one file at the same time:
  - Multiple exclusive locks
  - Shared locks and an exclusive lock

  Write permission is required to obtain an exclusive lock.

If you use the Clib Lock File method to:

- Lock a file that a calling process already locked, then it removes the old lock type and replaces it with the new lock type. The Lock method locks individual files and not segments.
- Lock a file that is already locked and:
  - You do not specify LOCK_NB in the mode argument, then it pauses the lock request until the file is free.
  - Specify LOCK_NB in the mode argument, then the call fails and this method returns an EWOULDBLOCK error.

Siebel eScript does not support the Clib Lock File method in a Unicode environment. It always returns 0 in a Unicode environment.

Clib Open File Method

The Clib Open File method opens the file that you specify in the filename argument. It opens it in the mode that you specify in the mode argument. It returns one of the following values:

- If successful, then it returns a file pointer to the file that it opened.
- If not successful, then it returns the following value:

  Null

If this method successfully opens a file, then it clears the error status for this file and initializes a buffer for automatic buffering of read and write activity with the file.

Several Clib methods require an argument named filePointer. It is often the return value of a Clib Open File call.

Format

`Clib.fopen(`*`filename, mode`*`)`

Table 127 describes the arguments for the Clib Open File method.

Table 127. Arguments for the Clib Open File Method

Argument	Description
filename	Any valid file name that does not include a wildcard character.
mode	One of the required characters that specify a file mode followed by optional characters. For more information, see "Usage for the Mode Argument" on page 222.

Usage for the Mode Argument

Table 128 describes usage for the mode argument. The mode argument is a string that includes one of the following required characters, and then followed by other optional characters:

- r
- w
- a

Table 128. Usage for the Mode Argument of the Clib Open File Method

Argument	Mode	Required
r	Opens the file for reading. The file must already exist.	Yes. You must include one of these arguments.
w	Opens the file for writing. If the file does not exist, then Siebel eScript creates the file.	
a	Opens the file in append mode.	

Table 128. Usage for the Mode Argument of the Clib Open File Method

Argument	Mode	Required
b	Opens the file in binary mode. If you do not specify b, then this method opens the file in text mode and performs an end-of-line translation.	No
t	Opens the file in text mode. For a non-ASCII character: ■ You use the u argument. ■ You do not use the t argument.	No
u	Opens the file in Unicode mode as UTF-16 or Little Endian. For example: `Clib.fopen(“filename.txt”, “rwu”)` You can use the u mode for ASCII and non-ASCII characters.	No
+	Opens the file for reading and writing.	No

Example 1

The following example opens the ReadMe text file for text mode reading and displays each line in that file:

```
var fp:File = Clib.fopen("ReadMe","rt");
if ( fp == null )
   TheApplication().RaiseErrorText("\aError opening file for reading.\n")
else
{
   while ( null != (line=Clib.fgets(fp)) )
   {
      Clib.fputs(line, stdout)
   }
}
Clib.fclose(fp);
```

Example 2

The following example opens a file, writes a string to that file, and then uses the default codepage to read the string from this file:

```
var oFile = Clib.fopen("myfile","rw");
if (null != oFile)
{
   var sHello = "Hello";
   var nLen = sHello.length;
   Clib.fputs(sHello, oFile);
   Clib.rewind(oFile);
   Clib.fgets (nLen, sHello);
}
```

Example 3

The following example opens a file, writes a string to this file, then uses Unicode to read the string from this file:

```
var oFile = Clib.fopen("myfile","rwu");
if (null != oFile)
{
   var sHello = "Hello";
   var nLen = sHello.length;
   Clib.fputs(sHello, oFile);
   Clib.rewind(oFile);
   Clib.fgets (nLen, sHello);
}
```

Example 4

The following example specifies a file path:

```
function WebApplet_ShowControl (ControlName, Property, Mode, &HTML)
{
if (ControlName == "GotoUrl")
   {
      var fp = Clib.fopen("c:\\test.txt","wt+");
      Clib.fputs("property = " + Property + "\n", fp);
      Clib.fputs("mode = " + Mode + "\n",fp);
      Clib.fputs("ORG HTML = " + HTML + "\n",fp);
      Clib.fclose(fp);
      HTML = "<td>New HTML code</td>";
   }
return(ContinueOperation);
```

Related Topics

For more information, see the following topics:

- “Clib Create Temporary File Method” on page 219
- “Clib Get Environment Variable Method” on page 290

Clib Rename File Method

The Clib Rename File method renames a file. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns the following value:

  Negative 1

Format

```
Clib.rename(oldName, newName)
```

Table 129 describes the arguments for the Clib Rename File method.

Table 129. Arguments for the Clib Rename File Method

Argument	Description
oldName	A string that contains the name of the file that this method renames. This name can be an absolute file name or a relative file name.
newName	A string that contains the new file name.

Clib Reopen File Method

The Clib Reopen File method closes the file associated with a file pointer. It then opens a file and associates it with the file pointer of the file that it closed. You can use it to redirect one of the predefined file handles to a file or from a file. These file handles include stdout, stderr, and stdin. It returns one of the following values:

- If successful, then it returns a copy of the old file pointer.
- If not successful, then it returns the following value:

  Null

Format

```
Clib.freopen(filename, mode, oldFilePointer)
```

Table 130 describes the arguments for the Clib Reopen File method.

Table 130. Arguments for the Clib Reopen File Method

Argument	Description
filename	The name of the file that this method opens.
mode	One of the file modes specified in Clib Open File method. For Unicode, you can use the same u flag that you can use in the Clib Open File method.
oldFilePointer	The file pointer to the file that the Clib Reopen File method closes and where it associates the file that you specify in the filename argument.

Example

The following example uses the same file pointer to write to two different files:

```
var oFile = Clib.fopen("c:\\temp\\firstfile","w");
if (oFile == null)
{
```

```
      TheApplication().RaiseErrorText("File not found.");
   }
   Clib.fprintf(oFile, "Writing to first file\n");
   Clib.freopen("c:\\temp\\secondfile", "w", oFile);
   if (oFile == null)
   {
      TheApplication().RaiseErrorText("File not found.");
   }
   Clib.fprintf(oFile, "Writing to second file\n");
   Clib.fclose(oFile);
```

Related Topics

For more information, see the following topics:

- "Clib Open File Method" on page 222
- "Clib Get Environment Variable Method" on page 290

Clib Change Directory Method

The Clib Change Directory method modifies the current directory for the Siebel application. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns the following value:

  Negative 1

If you restart the Siebel Server, then Siebel CRM automatically resets the current directory depending on one of the following operating systems that you use:

- **Windows.** The current directory on the Siebel Server that the Windows operating system recognizes.
- **UNIX.** The home directory of the administrator who restarts the Siebel Server.

Format

`Clib.chdir(`*`dirPath`*`)`

Table 131 describes the arguments for the Clib Change Directory method.

Table 131. Arguments for the Clib Change Directory Method

Argument	Description
dirpath	The directory path that this method makes current. This path can be absolute or relative.

Example

The following example uses the Clib Change Directory method to change the current working directory of the Siebel application. The default Siebel working directory is *SIEBEL_ROOT*\bin. For example, if you install the Siebel client in the C:\sea81\client directory, then the default working directory is C:\sea81\client\bin:

```
function Application_Start (CommandLine)
{

   // Start Tracing
   TheApplication().TraceOn("c:\\temp\\SiebTrace.txt","Allocation","All");

   var currDir = Clib.getcwd();
   TheApplication().Trace("Current directory is " + Clib.getcwd());

   // Create a new directory
   var msg = Clib.mkdir('C:\\Clib test');

   // Display the error flag created by creating directory;
   // Must be 0, indicating no error.

   TheApplication().Trace(msg);

   // Change the current directory to the new 'Clib test'
   Clib.chdir("C:\\Clib test");
   TheApplication().Trace("Current directory is " + Clib.getcwd());

   // Delete 'Clib test'
   Clib.chdir("C:\\");

   // Attempting to make a removed directory current gives an
   // error
   Clib.rmdir("Clib test");
   msg = Clib.chdir("C:\\Clib test");
   TheApplication().Trace(msg);
}
```

This example produces the following result:

```
Current directory is D:\sea81\client\BIN
0
Current directory is C:\Clib test
-1
```

Clib Create Directory Method

The Clib Create Directory method creates a directory. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns the following value:

  negative 1

Format

```
Clib.mkdir(dirpath)
```

Table 132 describes the arguments for the Clib Create Directory method.

Table 132. Method Arguments for the Clib Create Directory

Argument	Description
dirpath	A string that contains a valid directory path. This directory can be an absolute path or a relative path. This method uses this string to create the directory. If you do not specify the dirpath argument, then it creates the directory in the C:\siebel\bin directory.

Clib Get Current Working Directory Method

The Clib Get Current Working Directory method returns the entire path of the current working directory. The default current working directory is the directory where you install the Siebel application.

If a script uses the Clib Change Directory method or a similar method to change the current working directory, then the current working directory returns to the original value after the script finishes.

Format

```
Clib.getcwd()
```

Example

The following example displays the current directory in a message box. The script then makes the root directory the current directory, creates a new directory, removes that directory, and then attempts to make the removed directory current:

```
function Button_Click ()

{

   var currDir = Clib.getcwd();
   TheApplication().Trace("Current directory is " + Clib.getcwd());
   var msg = Clib.mkdir('C:\\Clib test');
   // Display the error flag created by creating directory;
   // Must be 0, indicating no error.
   TheApplication().Trace(msg);
   // Change the current directory to the new 'Clib test'
   Clib.chdir("C:\\Clib test");
```

```
   TheApplication().Trace("Current directory is " + Clib.getcwd());
   // Delete 'Clib test'
   Clib.chdir("C:\\");
   // Attempting to make a removed directory current yields error
       flag
   Clib.rmdir("Clib test");
   msg = Clib.chdir("C:\\Clib.test");
   TheApplication().Trace(msg);
}
```

This example displays the following output:

```
Current directory is C:\SIEBEL\BIN
0
Current directory is C:\Clib test
-1
```

Clib Remove Directory Method

The Clib Remove Directory method removes a directory. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns the following value:

  Negative 1

Format

`Clib.rmdir(`*`dirpath`*`)`

Table 133 describes the arguments for the Clib Remove Directory method.

Table 133. Arguments for the Clib Remove Directory Method

Argument	Description
dirpath	The directory that this method removes. This argument can reference an absolute path or a relative path.

Clib File Input and Output Methods

This topic describes Clib file input and output methods. It includes the following topics:

Overview of Clib File Input and Output Methods

Siebel eScript handles file input and file output operations in a way that is similar to the C programming language and the C++ programming language. These languages do not directly read to or write from files. With Siebel eScript, you must first configure the language to open a file. To do this, you typically pass the name of this file to the Clib Open File method.

File input and file output methods in Siebel eScript read the file into a buffer in memory and return a *file pointer*, which is a pointer that references the beginning of the buffer. The *file stream* is the data that the buffer contains. Reading and writing occurs relative to the buffer, which is not written to disk unless you explicitly use the Clib Clear Buffer method to clear the buffer or use the Clib Close File method to close the file.

Format Characters for Methods That Print and Scan

A method that prints or scans uses a format string to format the data that the method reads and writes.

Format Characters for Methods That Print

This topic describes format characters for methods that print. The following methods can perform print operations:

- “Clib Write Formatted String Method” on page 248

■ "Clib Get Formatted String Method" on page 255

■ "Clib Write Formatted String Method" on page 261

Each of these methods prints each character while it reads the input until the method encounters a percentage symbol (%). This symbol instructs that method to use the following format to print a value:

%[*flags*][*width*][.*precision*]*type*

To include the % symbol as a character in the string, you use two consecutive percentage symbols (%%).

Characters That Format Values

Table 134 describes characters that format a value.

Table 134. Characters That Format a Value

Character	Description	Example Statement and Output
-	Left justification in the field with space padding or right justification with zero or space padding.	`fprintf(file, "[%-8i]", 26);` `[26 ]`
+	Force numbers to begin with a plus symbol (+) or a minus symbol (-).	`fprintf(file, "%+i", 26);` `+26`
space	A negative value that begins with a minus symbol (-). A positive value begins with a space.	`fprintf(file, "[% i]", 26);` `[ 26]`

#	Append one of the following symbols to the pound (#) character to display the output in one of the following forms:
	■ **o.** Prefix a zero to nonzero octal output.
	■ **x or X.** Prefix 0x or 0X to the output, which indicates hexadecimal.
	■ **f.** Include a decimal point even if no digits follow the decimal point.
	■ **e or E.** Include a decimal point even if no digits follow the decimal point, and display the output in scientific notation.
	■ **g or G.** Include a decimal point even if no digits follow the decimal point, display the output in scientific notation, depending on precision, and leave trailing zeros in place.
		`fprintf(file, "%#o", 26);`
		`032`
		`fprintf(file, "%#x", 26);`
		`0x1A`
		`fprintf(file, "%#.f", 26);`
		`26.`
		`fprintf(file, "%#e", 26);`
		`2.600000e+001`
		`fprintf(file, "%#g", 26);`
		`26.0000`

Table 134. Characters That Format a Value

Character	Description	Example Statement and Output
f	Floating-point of the format [-]dddd.dddd.	`fprintf(file, "%f", 26.735);` `26.735000`
e	Floating-point of the format [-]d.ddde+dd or [-]d.ddde-dd.	`fprintf(file, "%e", 26.735);` `2.673500e+001`
E	Floating-point of the format [-]d.dddE+dd or [-]d.dddE-dd.	`fprintf(file, "%E", 26.735);` `2.673500E+001`
g	Floating-point number of f or e type, depending on precision.	`fprintf(file, "%g", 26.735);` `26.735`
G	Floating-point number of F or E type, depending on precision.	`fprintf(file, "%G", 26.735);` `26.735`
c	Character. For example, a, b, or 8.	`fprintf(file, "%c", 'a');` `a`
s	String.	`fprintf(file, "%s", "Test");` `Test`

Characters That Determine Width

Table 135 describes characters that determine width.

Table 135. Characters That Determine Width

Character	Description	Example Statement and Output
n	At least n characters are output. If the value is less than n characters, then Siebel eScript pads the output on the left with spaces.	`fprintf(file, "[%8s]", "Test");` `[ Test]`
0*n*	At least n characters are output, padded on the left with zeros.	`fprintf(file, "%08i", 26);` `00000026`
*	The next value in the argument list is an integer that specifies the output width.	`fprintf(file, "[%*s]", 8, "Test");` `[ Test]`

Characters That Determine Precision

Table 136 describes characters that determine precision. If you specify precision, then you must begin the precision format with a period (.) and you must use one of the forms described in Table 134 on page 231.

Table 136. Characters That Determine Precision

Character	Description	Example Statement and Output
.0	For floating-point type. No decimal point is output.	`fprintf(file, "%.0f", 26.735);` `26`

.n	Output is n characters. If the value is a floating-point number, then the output is n decimal places.
	
	Assume you specify a Width value and a *.n* Precision value when you format a floating point number. In this situation, to determine the width of the output and to determine if it must pad the output, the method counts the decimal point and the characters that occur before and after the decimal point. For example:
	
	`fprintf(file, "%10.2f", 26.735);`
	`[ 26.73]`
		`fprintf(file, "%.2f", 26.735);`
		`26.73`

.*	The next value in the argument list is an integer that specifies the precision width.	`fprintf(file, "%.*f", 1, 26.735);` `26.7`

Characters That Determine Character Type

Table 137 describes characters that determine character type.

Table 137. Characters That Determine Character Type

Character	Description	Example Statement and Output
d,i	Signed integer.	`fprintf(file, "%i", 26);` `26`
u	Unsigned integer.	`fprintf(file, "%u", -1);` `4294967295`
o	Octal integer.	`fprintf(file, "%o", 26);` `32`
x	Hexadecimal integer using 0 through 9 and a, b, c, d, e, or f.	`fprintf(file, "%x", 26);` `1a`
X	Hexadecimal integer using 0 through 9 and A, B, C, D, E, or F.	`fprintf(file, "%X", 26);` `1A`

Table 137. Characters That Determine Character Type

Character	Description	Example Statement and Output
f	Floating-point of the format [-]dddd.dddd.	`fprintf(file, "%f", 26.735);` `26.735000`
e	Floating-point of the format [-]d.ddde+dd or [-]d.ddde-dd.	`fprintf(file, "%e", 26.735);` `2.673500e+001`
E	Floating-point of the format [-]d.dddE+dd or [-]d.dddE-dd.	`fprintf(file, "%E", 26.735);` `2.673500E+001`
g	Floating-point number of f or e, depending on precision.	`fprintf(file, "%g", 26.735);` `26.735`
G	Floating-point number of F or E, depending on precision.	`fprintf(file, "%G", 26.735);` `26.735`
c	Character. For example, a, b, 8.	`fprintf(file, "%c", 'a');` `a`
s	String.	`fprintf(file, "%s", "Test");` `Test`

Format Characters for Methods That Scan

This topic describes format characters for methods that scan. The following methods can perform a scan operation:

- "Clib Scan and Convert File Method" on page 244
- "Clib Scan and Convert from Input Device Method" on page 246

Note the following:

- The format string includes character combinations that specify the type of data.
- The format string specifies input sequences and how the method must convert the input.
- The method maps each character to the input as it reads the input until it encounters a percentage symbol (%).
- The percentage symbol causes the method to read the value, and then store it in an argument that follows the format string.
- Each argument that occurs after the format string receives the next parsed value from the next argument in the list of arguments that occur after the format string.

Arguments In a Method That Performs a Scan Operation

An argument in a method that performs a scan operation uses the following format:

`%[*][`*`width`*`]`*`type`*

Table 138 describes usage of the * (asterisk) and the width argument. If you specify the width, then the input is an array of characters of the length that you specify.

Table 138. Usage of the Asterisk and Width Arguments in a Method That Performs a Scan Operation

Argument	Description
*	Suppresses assigning this value to any argument.
width	Sets the maximum number of characters to read. If the method encounters a white-space character or a nonconvertible character, then it stops reading these characters. For more information, see "Use White Space to Improve Readability" on page 56.

Table 139 describes the values you can use for the type argument.

Table 139. Usage of the Type Argument in a Method That Performs a Scan Operation

Type Value	Description
d,D,i,I	Signed integer.
u,U	Unsigned integer.
o,O	Octal integer.
x,X	Hexadecimal integer.
f,e,E,g,G	Floating-point number.
s	String.
[abc]	String that includes the characters in brackets, where A–Z represents the range A to Z.
[^abc]	String that includes the following character in brackets: not

Example

The following example creates a file named myfile.txt and stores a float number and a string. It then rewinds the stream and uses fscanf to read the values:

```
function WebApplet_Load()
{
   var f;
   var str;
   var pFile = Clib.fopen ("c:\\myfile.txt","w+");
   Clib.fprintf (pFile, "%f %s", 3.1416, "PI");
   Clib.rewind (pFile);
   Clib.fscanf (pFile, "%f", f);
   Clib.fscanf (pFile, "%s", str);
```

```
      Clib.fclose (pFile);
      Clib.printf ("I have read: %f and %s \n",f,str);
   }
```

This example produces the following output:

```
I have read: 3.141600 and PI
```

Clib Clear Buffer Method

The Clib Clear Buffer method writes to disk the data that exists in the buffer depending on the following value in the filePointer argument:

- **Is not null.** It writes to disk any data that exists in the buffer only for the file that the filePointer argument identifies.
- **Is null.** It writes to disk any data that exists in the buffer for all open files.

This method returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns the following value:

  EOF

Format

`Clib.fflush(`*`filePointer`*`)`

The arguments for this method are the same as the arguments for the Clib Clear Error method. For more information, see "Arguments for the Clib Clear Error Method" on page 286.

Related Topics

For more information, see "Clib Get Environment Variable Method" on page 290.

Clib End of File Method

The Clib End of File method determines if the file cursor is at the end of the file that the filePointer argument identifies. It returns one of the following values:

- If the file cursor is at the end of the file, then it returns the following value:

  A nonzero integer

- If the file cursor is not at the end of the file, then it returns the following value:

  0

Format

```
Clib.feof(filePointer)
```

The arguments for this method are the same as the arguments for the Clib Clear Error method. For more information, see "Arguments for the Clib Clear Error Method" on page 286.

Clib Get Character Method

The Clib Get Character method returns one of the following values:

- The next character from the buffer of the file that the filePointer argument identifies. It returns this value as a byte converted to an integer.
- If a read error occurs or if the cursor is at the end of the file, then it returns the following value and stores the error number in the errno property:

  EOF

Format

```
Clib.getc(filePointer)
Clib.fgetc(filePointer)
```

The arguments for these methods are the same as the arguments for the Clib Clear Error method. For more information, see "Arguments for the Clib Clear Error Method" on page 286.

In most situations, to avoid an error with macro usage, you must use Clib.fgetc.

Clib Get Characters to Next Line Method

The CLib Get Characters to Next Line method returns one of the following values:

- A string that includes the characters that exist in a file from the current position of the file cursor up to and including the next newline character.
- If an error occurs or if it reaches the end of the file, then it returns the following value:

  Null

Format

```
Clib.fgets([maxLen,] filePointer)
```

Table 140 describes the arguments for the Get Characters to Next Line method.

Table 140. Arguments for the Get Characters to Next Line Method

Argument	Description
maxLen	The maximum length of the string that this method returns if it does not encounter a newline character. If the File Mode is Unicode, then the maxLen argument is the length in Unicode characters. If you do not specify the maxLen argument, then Siebel eScript uses the default limit of 999 characters.
filePointer	A file pointer that the Clib Open File method returns.

Example

The following example writes a string that contains an embedded newline character to a temporary file. To return and display the output, it then reads from the file twice:

```
function Test_Click ()
{
   var x = Clib.tmpfile();
   Clib.fputs("abcdefg\nABCDEFG\n", x);
   Clib.rewind(x);
   var msg = Clib.fgets(x) + " " + Clib.fgets(x);
   Clib.fclose(x);
   TheApplication().RaiseErrorText(msg);
}
```

This example produces the following output:

```
abcdefg
ABCDEFG
```

Caution About Using the Get Characters to Next Line Method with Non-ASCII Characters

If the string that the Get Characters to Next Line method returns includes a non-ASCII character, then you must configure Siebel CRM to open in Unicode the file that the filePointer argument specifies.

CAUTION: If Siebel CRM opens the file in text mode, then this method treats any non-ASCII character it encounters as an end-of-line character and stops reading the current line. As a result, this method might truncate the string that it returns. If the file does not use an encoding standard that is compatible with Unicode, then you must first configure Siebel CRM to transform it to UTF-8 or UTF-16 with the appropriate byte-order mark (BOM) placed at the beginning of the file. For more information, see “Clib Open File Method” on page 222.

Related Topics

For more information, see “Clib Write String to File Method” on page 250.

Clib Get Cursor Position Method

The Clib Get Cursor Position method gets the current position of the file cursor in the file that the filePointer argument identifies. It stores this value in the position argument.

Format

`Clib.fgetpos(`*`filePointer, position`*`)`

Table 141 describes the arguments for the Get Pointer Position method.

Table 141. Arguments for the Clib Get Cursor Position Method

Argument	Description
filePointer	A file pointer that the Clib Open File method returns.
position	The current position of the pointer in the file that the filePointer argument identifies.

Example

The following example restores the cursor position. It does the following work:

1. Writes two strings to a temporary text file.
2. To save the position where the second string begins, it uses the Clib Get Cursor Position method.
3. To set the file cursor to the saved position, it uses the Clib Set Cursor Position method:

```
function Test_Click ()
{
   var position;
   var fp = Clib.tmpfile();
   Clib.fputs("Melody\n”, fp);
   Clib.fgetpos(fp, position)
   Clib.fputs("Lingers\n", fp);
   Clib.fsetpos(fp, position);
   var msg = Clib.fgets(fp));
   Clib.fclose(fp);
   TheApplication().RaiseErrorText(msg);
}
```

Clib Get Relative Cursor Position Method

The Clib Get Relative Cursor Position method gets the position of the file cursor of an open file relative to the beginning of the file. It returns one of the following values:

- If successful, then it returns the current position of the file cursor.

- If not successful, then it returns the following value and stores the error value in the errno property:

  Negative 1

The cursor position in a text file might not correspond exactly with the byte offset in the file. A *text file* is a file that is not opened in binary mode.

Format

`Clib.ftell(`*`filePointer`*`)`

The arguments for this method are the same as the arguments for the Clib Clear Error method. For more information, see "Arguments for the Clib Clear Error Method" on page 286.

Clib Move Cursor to Beginning of File Method

The Clib Move Cursor to Beginning of File method moves the file cursor to the beginning of a file. This method is identical to the Clib Set Cursor Position method with the mode argument set to SEEK_SET and the offset argument set to 0. The only difference is that the Clib Move Cursor to Beginning of File method also clears the error indicator for the file.

Format

`Clib.rewind(`*`filePointer`*`)`

The arguments for this method are the same as the arguments for the Clib Clear Error method. For more information, see "Arguments for the Clib Clear Error Method" on page 286.

Usage With a Unicode File

Siebel CRM uses UTF-16 encoding when it writes to a file in Unicode. The first two bytes of the file are always the BOM (Byte Order Mark). If the Clib Move Cursor to Beginning of File method calls a Unicode file, then it references BOM (-257) and not the first valid character. To skip the BOM, you must configure Siebel CRM to call the Clib Get Character Method or the Clib File Get Character Method method at least one time. For more information, see "Clib Get Character Method" on page 237.

Example

For an example, see "Clib Get Characters to Next Line Method" on page 237.

Clib Read From File Method

The Clib Read From File method reads data from an open file that you specify in the filePointer argument. It then stores this data in an argument, buffer, or BLOB that you specify. If this argument, buffer, or BLOB does not exist, then this method creates it. It returns one of the following values:

- If successful, then it returns the number of elements it read.

- If you specify the destBuffer argument, then it returns the number of bytes read, up to the value you specify in the bytelength argument.
- If you specify the varDescription argument, then it returns one of the following values:
  - 1 if it reads the data
  - 0 if a read error occurs or if it encounters the end of file

Format A

`Clib.fread(`*`destBuffer, bytelength, filePointer`*`)`

Format B

`Clib.fread(`*`destVar, varDescription, filePointer`*`)`

Format C

`Clib.fread(`*`blobVar, blobDescriptor, filePointer`*`)`

Arguments

Table 142 describes the arguments for the Clib Read From File method.

Table 142. Arguments for the Clib Read From File Method

Argument	Description
destBuffer	The buffer to contain the data that this method reads.
bytelength	The number of bytes that this method reads.
filePointer	A file pointer that the Clib Open File method returns.
destVar	A container to hold the data that this method reads.
varDescription	The format of the data that this method reads. For more information, see "Format of the Data That the Clib Read From File Method Reads" on page 242.
blobVar	The BLOB where this method writes data.
blobDescriptor	The BLOB descriptor for the value you specify in the blobVar argument.

Format of the Data That the Clib Read From File Method Reads

Table 143 describes the format of the data that the Clib Read From File method reads. You specify this format in the varDescription argument. If the destVar argument must hold a single datum, then you must set the varDescription argument to one of these formats. If the destVar contains blob data, then you must specify a blobdescriptor argument. A blobdescriptor can also consist of varDescriptions for the individual elements of the blobdescriptor.

Table 143. Format of the Data That the Clib Read From File Method Reads

Value	Description
UWORD8	Stored as an unsigned byte.
SWORD8	Stored as a signed byte.
UWORD16	Stored as an unsigned, 16-bit integer.
SWORD16	Stored as a signed, 16-bit integer.
UWORD24	Stored as an unsigned, 24-bit integer.
SWORD24	Stored as a signed, 24-bit integer.
UWORD32	Stored as an unsigned, 32-bit integer.
SWORD32	Stored as a signed, 32-bit integer.
FLOAT32	Stored as a floating-point number.
FLOAT64	Stored as a double-precision, floating-point number.

The following code includes example formats:

```
ClientDef = new blobDescriptor();
ClientDef.Sex = UWORD8;
ClientDef.MaritalStatus = UWORD8;
ClientDef._Unused1 = UWORD16;
ClientDef.FirstName = 30; ClientDef.LastName = 40;
ClientDef.Initial = UWORD8;
```

Usage for the Clib Read From File Method

The Siebel eScript usage of fread differs from the standard C library usage in that the C library reads an array of numeric values or structures into consecutive bytes in memory. The Clib Read From File method reads data in the byte-order that the current value of the BigEndianMode global variable describes.

Example

The following example reads the following items from the fp file:

- Reads the 16-bit i integer
- Reads the 32-bit f float
- Reads the 10-byte buffer from the buf buffer:

```
if ( !Clib.fread(i, SWORD16, fp) || !Clib.fread(f, FLOAT32, fp)
|| 10 != Clib.fread(buf, 10, fp) )
   TheApplication().RaiseErrorText("Error reading from file.\n");
}
```

Clib Restore Cursor Position Method

The Restore Cursor Position method sets the current file cursor to a position that you specify. You can use it to restore the file cursor to a position that the Clib Get Cursor Position returns. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns nonzero and stores the error value in the errno property.

Format

`Clib.fsetpos(`*`filePointer, position`*`)`

Table 144 describes the arguments for the Restore Cursor Position method.

Table 144. Arguments for the Restore Cursor Position Method

Argument	Description
filePointer	A file pointer that the Clib Open File method returns.
position	The value that the Clib Get method returns.

Example

For an example, see “Clib Get Cursor Position Method” on page 239.

Related Topics

For more information, see the following topics:

- “Clib Get Cursor Position Method” on page 239
- “Clib Get Relative Cursor Position Method”

Clib Set Cursor Position Method

The Clib Set Cursor Position method sets the position of the file cursor of an open file. It returns one of the following values:

- If successful, then it returns the following value:

  0

- If not successful, then it returns a nonzero value.

Format

```
Clib.fseek(filePointer, offset[, mode])
```

Table 145 describes the arguments for the Clib Set Cursor Position method.

Table 145. Arguments for the Clib Set Cursor Position Method

Argument	Description
filePointer	A file pointer that the Clib Open File method returns.
offset	The number of bytes that the Clib Set Cursor Position method moves the file cursor, starting with the value that you specify in the mode argument. The cursor position in a text file might not correspond exactly with the byte offset in the file. A text file is a file that is not opened in binary mode.
mode	You can specify one of the following values: ■ **SEEK_CUR.** Relative to the current position of the file cursor. ■ **SEEK_END.** Relative to the end of the file. ■ **SEEK_SET.** Relative to the beginning of the file. If you do not specify the mode argument, then this method uses SEEK_SET.

Related Topics

For more information, see the following topics:

- "Clib Get Cursor Position Method" on page 239
- "Clib Get Relative Cursor Position Method" on page 239
- "Clib Move Cursor to Beginning of File Method" on page 240

Clib Scan and Convert File Method

The Clib Scan and Convert File method reads data from a file and stores data items that exist in this file in a series of arguments. It returns one of the following values:

- If successful, then it returns the number of input items it converted and stored.
- If an input failure occurs before the conversion, then it returns the following value:

  EOF

Format

```
Clib.fscanf(filePointer, formatString, var1, var2, ..., varn)
```

Table 146 describes the arguments for the Clib Scan and Convert File method.

Table 146. Arguments for the Clib Scan and Convert File Method

Argument	Description
filePointer	A file pointer that the Clib Open File method returns.
formatString	A string that contains format instructions that the Clib Open File method uses to read each data item in the file.
var1, var2, ..., varn	Variables that the Clib Open File method uses to store the values that it formats.

Usage

This method does the following work:

1. Reads input from the file that the filePointer argument identifies.
2. Matches characters that exist in the file with characters that the formatString argument specifies until it reaches a percentage symbol (%).

   The percentage symbol causes this method to read and store the values in the arguments that occur after the string that the formatString argument identifies.
3. Parses each match that occurs after the value of the formatString argument.

   As it parses each match, it stores the result in a variable argument, such as var1, var2, ..., and varn. If a matching failure occurs, then the number of matches it parses might be fewer than the number of variable arguments you specify.

An argument specification uses the following format:

```
%[*][width]type
```

For values for these items, see "Format Characters for Methods That Scan" on page 234.

You must make sure that the file it reads is open and includes read access.

Example

The following example uses the Clib Scan and Convert File method with various options on the arguments:

```
var int1;
var int2;
var hour;
var min;
var sec;
var str;

var file = Clib.fopen("c:\\temp\\fscanf.txt", "r");
TheApplication().TraceOn("c:\\temp\\testoutput.txt", "allocation", "all");

// Simple scanf:
```

```
// input line e.g.: "Monday 10:18:00"
Clib.fscanf(file, "%s %i:%i:%i\n", str, hour, min, sec);
TheApplication().Trace(str + ", " + hour + ", " + min + ", " + sec);

// Using width specifier:
// input line e.g.: "1234567890"
Clib.fscanf(file, "%5i%5i\n", int1, int2);
TheApplication().Trace(int1 + ", " + int2);

// Reading hexadecimal integers and suppressing assignment to a variable:
// input line e.g.: "AB3F 456A 7B44"
Clib.fscanf(file, "%x %*x %x\n", int1, int2);
TheApplication().Trace(int1 + ", " + int2);

// Using character ranges:
// input line e.g.: "helloHELLO"
Clib.fscanf(file, "%[a-z]\n", str);
TheApplication().Trace(str);

Clib.fclose(file);
```

This example produces the following output:

```
COMMENT,"Monday, 10, 18, 0"
COMMENT,"12345, 67890"
COMMENT,"43839, 31556"
COMMENT,hello
```

Clib Scan and Convert from Input Device Method

The Clib Scan and Convert from Input Device method reads input from an input device and stores the data in arguments. It reads from the keyboard unless the Clib Reopen File method redirects it to another file as stdin. It returns one of the following values:

- If successful, then it returns the number of variables where it assigned data.
- If not successful, then it returns the following value:

  EOF

This method does not read the input until the user presses the ENTER key. This method is identical to the Clib Scan and Convert File method with stdin set as the first argument. For more information, see “Clib Scan and Convert File Method” on page 244.

Format

Clib.sscanf([*formatString*] [,*var1, var2, ..., varn*])

Clib Unget Method

The Clib Unget method pushes a character back into a file. It returns one of the following values:

- If successful, then it returns the value that the char argument contains.
- If not successful, then it returns the following value:

  EOF

If this method pushes a character back into a file, then it converts the character that you specify in the char argument to a byte. It only pushes back one character. After the unget, this character is again available in the file for subsequent retrieval. You might need to use this method to read up to, but not including, a newline character. You can then use it to push the newline character back into the file buffer.

Format

```
Clib.ungetc(char, filePointer)
```

Table 147 describes the arguments for the Clib Unget method.

Table 147. Arguments for the Clib Unget Method

Argument	Description
char	The character that this method pushes back. It puts back one character to the file stream that it reads. It moves the seek position of the file pointer by one character position.
filePointer	A file pointer that the Clib Open File method returns.

Clib Write Character Method

The Clib Write Character method writes a character, converted to a byte, to a file that you specify. It returns one of the following values:

- If successful, then it returns the value that the char argument contains.
- If not successful, then it returns the following value:

  EOF

The following type of character that the char argument contains determines how this method writes the character:

- **String.** It writes the first character of the string to the file.
- **Number.** It writes the character that corresponds to the Unicode value for this number to the file.

Format

```
Clib.fputc(char, filePointer)
Clib.putc(char, filePointer)
```

Clib.fputc writes a character to a file. Clib.putc writes a character to the screen. In most situations, to avoid an error with macro usage, you must use Clib.fputc.

Table 148 describes the arguments for the Clib Write Character method.

Table 148. Arguments for the Clib Write Character Method

Argument	Description
char	A one character string or variable that contains a single character.
filePointer	A file pointer that the Clib Open File method returns.

Clib Write Formatted String Method

The Clib Write Formatted String method writes a formatted string to a file.

Format

`Clib.fprintf(`*`filePointer, formatString`*`)`

Table 149 describes the arguments for the Clib Write Formatted String method.

Table 149. Arguments for the Clib Write Formatted String Method

Argument	Description
filePointer	A file pointer that the Clib Open File method returns.
formatString	A string that contains formatting instructions for each data item that the Clib Write Formatted String method writes. For more information, “Format Characters for Methods That Print and Scan” on page 230.

Example

The following example uses the Clib Write Formatted String method with various values for the formatString argument:

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
   if (MethodName == "fprintfsamples")
   {
      var intgr = 123456789;
      var flt = 12345.6789;
      var hour = 1;
      var min = 7;
      var sec = 0;
      var str = "Hello World";
      var file = Clib.fopen("c:\\temp\\fprintf.txt", "w");

      // Simple formatting:
```

```
        Clib.fprintf(file, "(1) %s, it is now %i:%i:%i pm.\n", str, hour, min, sec);
        Clib.fprintf(file, "(2) The number %i is the same as %x.\n", intgr, intgr);
        Clib.fprintf(file, "(3) The result is %f.\n", flt);

        // Flag values:
        // "+" forces a + or - sign; "#" modifies the type flag "x"

        // to prepend "0x" to the output. (Compare with the simple
        // formatting example.)
        Clib.fprintf(file, "(4) The number %+i is the same as %#x.\n", intgr, intgr);

        // Width values:
        // The width is a minimal width, thus longer values
        // are not truncated.
        // "2" fills with spaces, "02" fills with zeros.
        var myWidth = 2;
        Clib.fprintf(file, "(5) %5s, it is now %2i:%02i:%02i pm.\n", str, hour, min,
        sec);

        // Precision values:
        // ".2" restricts to 2 decimals after the decimal separator.
        // The number will be rounded appropriately.
        Clib.fprintf(file, "(6) The result is %.2f.\n", flt);

        // A combined example:
        // <space> displays  space or minus;
        // "+" displays  plus or minus;
        // "020" uses a minimal width of 20, padded with zeros;
        // ".2" displays 2 digits after the decimal separator;
        // "*" uses the next argument in the list to specify the width.
        Clib.fprintf(file, "(7) The values are:\n%+020.2f\n% 020.2f\n% *.2f", flt,
        intgr, 20, intgr);

        Clib.fclose(file);

        return (CancelOperation);
      }
      return (ContinueOperation);
   }
```

This example produces the following output:

```
(1) Hello World, it is now 1:7:0 pm.
(2) The number 123456789 is the same as 75bcd15.
(3) The result is 12345.678900.
(4) The number +123456789 is the same as 0x75bcd15.
(5) Hello World, it is now  1:07:00 pm.
(6) The result is 12345.68.
(7) The values are:
+0000000000012345.68
0000000123456789.00
123456789.00
```

Clib Write String to File Method

The Write String to File method writes a string to a file that you specify. It returns one of the following values:

- If successful, then it returns a nonnegative value.
- If not successful, then it returns the following value:

  EOF

Format

```
Clib.fputs(string, filePointer)
```

Table 150 describes the arguments for the Write String to File method.

Table 150. Arguments for the Write String to File Method

Argument	Description
string	A string literal or a variable that contains a string.
filePointer	A file pointer that the Clib Open File method returns.

Example

For an example, see "Clib Get Characters to Next Line Method" on page 237.

Clib Write to File Method

The Clib Write to File method writes data to a file. It returns one of the following values:

- If successful, then it returns the number of elements it wrote
- If not successful, then it returns the following value:

  0

Siebel eScript usage of fwrite differs from the standard C library usage. The C library writes arrays of numeric values or structures from consecutive bytes in memory. This is not necessarily true in Siebel eScript.

Format A

```
Clib.fwrite(sourceVar, varDescription, filePointer)
```

Format B

```
Clib.fwrite(sourceVar, bytelength, filePointer)
```

Arguments

Table 151 describes the arguments for the Clib Write to File method.

Table 151. Arguments for the Clib Write to File Method

Argument	Description
bytelength	Number of bytes that this method writes.
sourceVar	The source that this method uses to get the data that it writes.
varDescription	A value that depends on the type of object that the sourceVar argument identifies.
filePointer	The file where this method writes data.

Usage for the varDescription Argument

Table 152 describes values you must set for the sourceVar argument and the varDescription argument. For example, if you use the sourceVar argument to identify a buffer, then you must set the varDescription argument to the length of that buffer, in bytes.

Table 152. Relative Values of the varDescription and sourceVar Arguments

Value of the sourceVar Argument	Value of the varDescription Argument
Buffer	Length of the buffer, in bytes.
Object	Value of the object descriptor.
A single datum	One of the values listed in “Format of the Data That the Clib Read From File Method Reads” on page 242.

Example

The following example writes the following data into the fp file:

- The 16-bit i integer
- The 32-bit f float
- The 10-byte buf buffer:

```
if ( !Clib.fwrite(i, SWORD16, fp) || !Clib.fwrite(f, FLOAT32, fp)
            || 10 !=  fwrite(buf, 10, fp))
{
   TheApplication().RaiseErrorText("Error writing to file.\n");
}
```

Clib String Methods

This topic describes Clib string methods. It includes the following topics:

Clib Append String Method

The Clib Append String method copies characters from one string to the end of another string. It appends up to the value that you specify in the maxLen argument of the string that you specify in the sourceString argument. It does not copy any character that occurs after a null byte. It returns the appended string that the destString argument contains.

The length of the destString argument is the lesser of the maxLen argument or the length of the sourceString argument.

Format

`Clib.strncat(`*`destString, sourceString, maxLen`*`)`

Table 153 describes the arguments for the Clib Append String method.

Table 153. Arguments for the Clib Append String Method

Argument	Description
sourceString	The string that this method uses to get the characters that it adds.
destString	The string where this method adds characters.
maxLen	The maximum number of characters to add.

Example

The following example uses the Clib Append String method:

```
var string1 = "I love to ";
var string2 = "ride hang-gliders and motor scooters.";
Clib.strncat(string1, string2, 17);
TheApplication().RaiseErrorText(string1);
```

This example returns the following string:

```
"I love to ride hang-gliders"
```

Related Topics

For more information, see "Clib Copy String Method" on page 254.

Clib Compare Strings Method

The Clib Compare Strings method performs a comparison between two strings, one byte at a time. It returns one of the following values:

- If the strings are identical, then it returns the following value:

  0

- If the ASCII code of the first unmatched character in the string1 argument is:
  - Less than that of the first unmatched character in the string2 argument, then it returns a negative number.
  - Greater than that of the first unmatched character in the string2 argument, then it returns a positive number.

It stops the comparison if one of the following situations occurs:

- It encounters a mismatch between strings.
- It encounters a terminating null byte.

Format

```
Clib.stricmp(string1, string2)
Clib.strcmpi(string1, string2)
Clib.strncmp(string1, string2, maxLen)
Clib.strncmpi(string1, string2, maxLen)
Clib.strnicmp(string1, string2, maxLen)
```

You can use one of the following:

- **Search that is case-sensitive.** You use Clib.strncmp.
- **Search that is not case-sensitive.** You use Clib.stricmp or Clib.strcmpi. In a comparison that is not case-sensitive, A and a are the same.

The Clib.strncmp, Clib.strncmpi, and Clib.strnicmp methods stop the comparison when one of the following situations occurs:

- It has compared the number of bytes that you specify in the maxLen argument.

- It encounters a terminating null byte.

Table 154 describes the arguments for the Clib Compare Strings method.

Table 154. Arguments for the Clib Compare Strings Method

Argument	Description
string1	A string or a variable that contains a string that this method compares against the string that the string2 argument contains.
string2	A string or a variable that contains a string that this method compares against the string that the string1 argument contains.
maxLen	The number of bytes to compare.

Clib Convert String to Lowercase Method

The Clib Convert String to Lowercase method converts a string to lowercase. It starts at position 0 of the str argument and ends immediately before the terminating null byte. It returns the value of the str argument all in lowercase.

Format

`Clib.strlwr(`*`str`*`)`

Table 155 describes the arguments for the Clib Convert String to Lowercase method.

Table 155. Arguments for the Clib Convert String to Lowercase Method

Argument	Description
str	The string that this method modifies to lowercase.

Clib Copy String Method

The Clib Copy String method copies characters from one string to another string. It returns the ASCII code of the first character of the string that you specify in the destString argument. You can write code that copies from one part of a string to another part of the same string.

Format

`Clib.strncpy(`*`destString, sourceString, maxLen`*`)`

This method uses the same arguments as the Clib Append String method. For more information, see "Clib Append String Method" on page 252. Note the following differences that the Clib Copy String method performs:

- The number of characters it copies is the lesser of the value of the maxLen argument and the length of the sourceString argument.
- If the value that the MaxLen argument contains is greater than the length of the value that the sourceString argument contains, then it fills the remainder of the destination string with null bytes.
- If the string you specify in the destString argument is not defined, then it defines this string.

Clib Get Formatted String Method

The Clib Get Formatted String method returns a formatted string as a numeric literal or as an argument.

If you use this method to format a floating point number to a specific number of decimal points, then it returns the value rounded to the number of decimal points that you specify. For example, if you use the following code to format the num argument, then it returns the num argument rounded to 2 decimal points:

```
Clib.rsprintf("%.2f", num)
```

Format

Clib.rsprintf([*formatString*] [,v*ar1, var2, ..., var*n])

Table 156 describes the arguments for the Clib Get Formatted String method.

Table 156. Arguments for the Clib Get Formatted String Method

Argument	Description
formatString	A string that includes character combinations that describe how to treat arguments. For more information on the format strings you can use with this method, see "Format Characters for Methods That Print and Scan" on page 230.
var1, var2, ..., varn	Variables that this method formats according to the format that you define in the formatString argument.

Example

Each of the following code lines includes an example of using the Clib Get Formatted String method followed by the resulting string:

```
var TempStr = Clib.rsprintf("I count: %d %d %d.",1,2,3) //"I count: 1 2 3"
var a = 1;
var b = 2;
TempStr = Clib.rsprintf("%d %d %d",a, b, a+b) //"1 2 3"
```

Clib Get Last Substring Method

The Clib Get Last Substring method searches a string for the last occurrence of a character. It returns one of the following values:

- If it finds the character, then it returns a string that includes the following items:
  - Begins at the rightmost occurrence of the value that you specify in the char argument
  - Ends with the rightmost character of the string that you specify in the string argument
- If it does not find the character, then it returns the following value:

  Null

It is recommended that you use the Clib Get Last Substring method only if you cannot use the equivalent standard JavaScript method.

Format

`Clib.strrchr(`*`string, char`*`)`

Table 157 describes the arguments for the Clib Get Last Substring method.

Table 157. Arguments for the Clib Get Last Substring Method

Argument	Description
string	A string literal or string variable that contains the character that this method searches.
char	The character that this method searches for.

Example

The following example uses the Clib Get Last Substring method:

```
var str = "I don’t like soggy cereal."
var substr = Clib.strrchr(str, 'o');
TheApplication().RaiseErrorText("str = " + str + "\nsubstr = " + substr);
```

This example provides the following result:

```
str = I don’t like soggy cereal.
substr = oggy cereal.
```

Related Topics

For more information, see “Create String From Substring Method” on page 90.

Clib Get Substring Method

The Clib Get Substring method searches a string for the first occurrence of a string. It returns one of the following values:

- If it finds the string that you specify in the findString argument, then it returns the string that:
  - Begins at the first occurrence of the value that you specify in the findString argument.
  - Ends at the end of the string that you specify in the sourceString argument.
- If it does not find the string that you specify in the findString argument, then it returns the following value:

  Null

It searches the string that you specify in the sourceString argument from the beginning of this string.

It is recommended that use the Clib Get Substring method only if you cannot use the equivalent standard JavaScript method.

Format

```
Clib.strstr(sourceString, findString)
Clib.strstri(sourceString, findString)
```

You can use one of the following:

- **Search that is case-sensitive.** You use Clib.strstr.
- **Search that is not case-sensitive.** You use Clib.strstri.

Table 158 describes the arguments for the Clib Get Substring method.

Table 158. Arguments for the Clib Get Substring Method

Argument	Description
sourceString	The string that this method searches.
findString	The string that this method must find.

Example 1

The following example uses Clib.strstr:

```
function Test1_Click ()
{
   var str = "We have to go to Haverford."
   var substr = Clib.strstr(str, 'H');
   TheApplication().RaiseErrorText("str = " + str + "\nsubstr = " +substr);
}
```

This example provides the following result:

```
str = We have to go to Haverford
substr = Haverford
```

Example 2

The following example uses Clib.strstri:

```
function Test_Click ()
{
   var str = "We have to go to Haverford."
   var substr = Clib.strstri(str, 'H');
   TheApplication().RaiseErrorText("str = " + str + "\nsubstr = " +substr);
}
```

This example provides the following result:

```
str = We have to go to Haverford.
substr = have to go to Haverford.
```

Related Topics

For more information, see "Create String From Substring Method" on page 90.

Clib Search String for Character Method

The Clib Search String for Character method searches a string for a character that you specify. It returns one of the following values:

- If it finds the character, then it returns the offset of the first occurrence of the character that you specify in the char argument. This offset is the number of characters in the string from the beginning to the first occurrence, starting with 0.
- If it does not find the character, then it returns the following value:

  Null

It is recommended that you use the Clib Search String for Character method only if you cannot use the equivalent standard JavaScript method.

Format

`Clib.strchr(`*`string, char`*`)`

Table 159 describes the arguments for the Clib Search String for Character method.

Table 159. Arguments for the Clib Search String for Character Method

Argument	Description
string	A string literal or a string variable that contains the character for which this method searches.
char	The character for which this method searches.

Example

The following example uses the Clib Search String for Character method:

```
var str = "I can't stand soggy cereal."
var substr = Clib.strchr(str, 's');
TheApplication().RaiseErrorText("str = " + str + "\nsubstr = " + substr);
```

This example products the following results:

```
I can't stand soggy cereal.
stand soggy cereal.
```

Clib Search String for Character Set Method

The Clib Search String for Character Set method searches a string for a set of characters that you specify in the charSet argument. It returns one of the following values:

- If it finds this set, then it returns the offset of the first character of the first occurrence of the set that you specify in the charSet argument. This offset is the number of characters in the string from the beginning to the first occurrence, starting with 0.
- If it does not find this set, then it returns the length of the string.

Format

`Clib.strcspn(`*`string, charSet`*`)`
`Clib.strpbrk(`*`string, charSet`*`)`

Clib.strcspn is similar to Clib.strpbrk, except that Clib.strpbrk returns the set that begins at the first character found while Clib.strcspn returns the offset number for that character.

Table 160 describes the arguments for the Clib Search String for Character Set method.

Table 160. Arguments for the Clib Search String for Character Set Method

Argument	Description
string	A literal string or a variable that contains the character set for which this method searches.
charSet	A literal string or a variable that is the character set for which this method searches.

Usage for the Clib Search String for Character Set Method

The Clib Search String for Character Set method searches for characters starting at the beginning of the string that you specify in the string argument. The search is case-sensitive, so you must use uppercase and lowercase characters in the charSet argument.

It is recommended that you use the Clib Search String for Character Set Method method only if you cannot use the equivalent standard JavaScript method.

Example

The following example demonstrates the difference between Clib.strcspn and Clib.strpbrk:

```
var string = "There's more than one way to climb a mountain.";
var rStrpbrk = Clib.strpbrk(string, "dxb8w9k!");
var rStrcspn = Clib.strcspn(string, "dxb8w9k!");
TheApplication().RaiseErrorText("The string is: " +  string +
   "\nstrpbrk returns a string: " +  rStrpbrk +
   "\nstrcspn returns an integer: " +  rStrcspn);
```

This example provides the following results:

```
The string is: There’s more than one way to climb a mountain.
strpbrk returns a string: way to climb a mountain.
strcspn returns an integer: 22
```

Clib Search String for Not Character Set Method

The Clib Search String for Not Character Set method searches a string for a set of characters that is not part of the value that you specify in the charSet argument. It returns one of the following values:

- If it finds all characters of the string that you specify in the charSet argument, then it returns the length of the string.
- If it does not find all characters of the string that you specify in the string argument, then it returns the offset of the first character that is not a member of the character set that you specify in the charSet argument.

Format

`Clib.strspn(`*`string, charSet`*`)`

This method uses the same arguments as the Clib Search String for Character Set method. Usage is also the same. For more information, see the following topics:

- “Clib Search String for Character Set Method” on page 259
- “Usage for the Clib Search String for Character Set Method” on page 259

Example

The following example searches for the value in the string argument, and then returns the position of w, counting from 0:

```
var string = "There is more than one way to swim.";
var rStrspn = Clib.strspn(string, " aeiouTthrsmn");
TheApplication().RaiseErrorText("strspn returns an integer: " + rStrspn);
```

This example provides the following results:

```
strspn returns an integer: 23
```

Clib Write Formatted String Method

The Clib Write Formatted String method writes output to a string variable according to a format that you define. It returns one of the following values:

- If successful, then it returns the number of characters it wrote in the buffer.
- If not successful, then it returns the following value:

  EOF

You are not required to define the string value. It is large enough to hold the result.

Format

Clib.sprintf(s*tringVar, formatString, var1, var2, ..., var*n)

This method performs the same work and uses the same arguments as the Clib Formatted String method except it also includes the stringVar argument. This argument identifies the name of the variable where the Clib Write Formatted String method writes the formatted string. For more information, "Clib Get Formatted String Method" on page 255.

Example

The following example uses the Clib Write Formatted String method with various format string arguments:

```
TheApplication().TraceOn("c:\\eScript_trace.txt","allocation","all");

var a, b, c;
a = 5;
b = 2;

Clib.sprintf(c, "First # %d + Second # %d is equal to %03d",a,b, a+b);
TheApplication().Trace("Output : " + c);

Clib.sprintf(c, "\n First # %d \n Second # %d \n => %d",12,16, 12+16)
TheApplication().Trace("Output : " + c);

var x, y, z, n;
var x = "Ali is 25 years old";
var y = "he lives in Ireland.";
var n = Clib.sprintf(z, "\n %s and %s",x,y)  ;

TheApplication().Trace("Output : " + z);
TheApplication().Trace("Total characters: " + n);

var  a = 16.51;
var  b = 5.79;
var c;

Clib.sprintf(c, "%.3f / %.3f is equal to %0.3f",a,b, parseFloat(a/b));
```

```
TheApplication().Trace("Output : " + c);

TheApplication().TraceOff();
```

This example produces the following result:

```
02/18/04,18:37:35,START,7.5.3 [16157] LANG_INDEPENDENT,SADMIN,3964,3836
02/18/04,18:37:35,COMMENT,Output : First # 5 + Second # 2 is equal to 007
02/18/04,18:37:35,COMMENT,"Output :
 First # 12
 Second # 16
 => 28"
02/18/04,18:37:35,COMMENT,"Output :
 Ali is 25 years old and he lives in Ireland."
02/18/04,18:37:35,COMMENT,Total characters: 46
02/18/04,18:37:35,COMMENT,Output : 16.510 + 5.790 is equal to 2.851
02/18/04,18:37:35,STOP
```

Clib Buffer Methods

This topic describes Clib buffer methods. It includes the following topics:

Clib Get Memory Method

The Clib Get Memory method searches a buffer for the first occurrence of a character that you specify. It returns one of the following values:

- If it finds the character you specify, then it returns the contents of the buffer starting at that character.
- If it does not find the character you specify, then it returns the following value:

  Null

Format

Clib.memchr*(bufferVar, char*[*, size*])

Table 161 describes the arguments for the Clib Get Memory method.

Table 161. Arguments for the Clib Get Memory Method

Argument	Description
bufferVar	A buffer or a variable that references a buffer.
char	The character that this method attempts to locate.
size	The number of bytes of the buffer that this method searches. It does one of the following depending on if you specify a size: ■ **You do specify a size.** It searches at the beginning of the buffer and continues until it reaches the point in the buffer that you indicate in the size argument. For example, if you specify the size as 1024, then it searches the first 1024 bytes of the buffer. ■ **You do not specify a size.** It searches the entire buffer from the first byte.

Clib Compare Memory Method

The Clib Compare Memory method compares the contents of two buffers. It returns one of the following values:

- If the value in the buf1 argument is less than the value in the buf2 argument, then it returns a negative number.
- If the value in the buf1 argument is greater than the value in the buf2 argument, then it returns a positive number.
- If the value in the buf1 argument is the same as the value in the buf2 argument, then it returns 0.

Format

```
Clib.memcmp(buf1, buf2[, length])
```

Table 162 describes the arguments for the Clib Compare Memory method.

Table 162. Arguments for the Clib Compare Memory Method

Argument	Description
buf1	A variable that contains the name of a buffer.

Table 162. Arguments for the Clib Compare Memory Method

Argument	Description
buf2	A variable that contains the name of a buffer.
length	The number of bytes that this method compares. It does one of the following depending on how you specify the length argument: ■ **You specify the length argument.** It compares the buffers from the first byte up to the length that you specify. For example, if you specify 1024 as the length, then it compares the first 1024 bytes of the buffer. ■ **You do not specify the length argument.** It compares the full length of the buffers. If one buffer is shorter than the other buffer, then it compares the buffers from the beginning byte up to the length of the shorter buffer.

Clib Copy Memory Method

The Clib Copy Memory method copies bytes from a source buffer to a destination buffer.

Format

```
Clib.memcpy(destBuf, srcBuf[, length])
Clib.memmove(destBuf, srcBuf[, length])
```

Siebel eScript protects data from being overwritten, so Clib.memmove performs exactly the same work as Clib.memcpy.

Table 163 describes the arguments for the Clib Copy Memory method.

Table 163. Arguments for the Clib Copy Memory Method

Argument	Description
destBuf	The name of a buffer or a variable that references a buffer. If this buffer does not exist, then this method creates it.
srcBuf	The buffer that this method uses to get the data that it copies.
length	The number of bytes that this method copies. If you do not specify the length argument, then it copies the entire contents of the buffer.

Clib Set Memory Method

The Clib Set Memory method sets the bytes in a buffer to a character that you specify.

Format

```
Clib.memset(bufferVar, char[, length])
```

Table 164 describes the arguments for the Clib Set Memory method.

Table 164. Arguments for the Clib Set Memory Method

Argument	Description
bufferVar	The name of a buffer or a variable that references a buffer. If this buffer does not exist, then this method creates it.
char	The character to which this method sets the bytes of the buffer.
length	The number of bytes that this method writes. This method does one of the following: ■ If the buffer is shorter than the value you specify in the length argument, then it increases the size of this buffer so that the size is equal to the value in the length argument. ■ If you do not specify the length argument, then it sets the length argument to the size of the buffer, starting at position 0.

Clib Mathematical Methods

This topic describes Clib mathematical methods. It includes the following topics:

Clib Create Random Number Method

The Clib Create Random Number method creates a pseudo-random number between 0 and RAND_MAX, inclusive. The value of RAND_MAX depends on the operating system. It is typically 32,768.

The initial value of the random number generator and earlier calls to the Clib Create Random Number method affects the sequence of pseudo-random numbers. For more information, see "Clib Initialize Random Number Generator Method" on page 270.

Format

```
Clib.rand()
```

Related Topics

For more information, see "Get Random Number Method" on page 189.

Clib Divide Method

The Clib Divide method performs integer division and returns a quotient and remainder.

Format

```
Clib.div(numerator, denominator)
Clib.ldiv(numerator, denominator)
```

Siebel eScript does not distinguish between integers and long integers, so clib.div and clib.ldiv are identical.

Table 165 describes the arguments for the Clib Divide method.

Table 165. Arguments for the Clib Divide Method

Argument	Description
numerator	The number that this method divides.
denominator	The number by which this method divides the numerator.

Table 166 describes the structure of the return value.

Table 166. Elements That the Clib Divide Method Returns

Element	Description
.quot	quotient
.rem	remainder

Example

The following example accepts two numbers as input from the user, divides the first number by the second number, and then displays the result:

```
var division = Clib.div(ToNumber(n), ToNumber(d));
   TheApplication().RaiseErrorText("The quotient is " + division.quot + ".\n\n" +
"The remainder is " + division.rem + ".");
```

If run this example with the values of n=9 and d=4, then it produces the following result:

```
The quotient is 2.

The remainder is 1.
```

Clib Get Floating Point Number Method

The Clib Get Floating Point Number method calculates a floating-point number given a mantissa and an exponent. It returns the result of the calculation. It calculates a floating-point number from the following equation:

mantissa multiplied by 2 ^ exponent

This method is the inverse of the Get Normalized Mantissa method. For more information, see "Clib Get Normalized Mantissa Method" on page 269.

Format

`Clib.ldexp(`*`mantissa, exponent`*`)`

Table 167 describes the arguments for the Clib Get Floating Point Number method.

Table 167. Arguments for the Clib Get Floating Point Number Method

Argument	Description
mantissa	The number on which this method operates.
exponent	The exponent that this method uses.

Clib Get Hyperbolic Cosine Method

The Clib Get Hyperbolic Cosine method calculates and returns the hyperbolic cosine of x.

Format

`Clib.cosh(`*`number`*`)`

Table 168 describes the arguments for the Clib Get Hyperbolic Cosine method.

Table 168. Arguments for the Clib Get Hyperbolic Cosine Method

Argument	Description
number	The hyperbolic cosine of the number that this method returns.

Clib Get Hyperbolic Sine Method

The Clib Get Hyperbolic Sine method calculates and returns the hyperbolic sine of a floating point number.

Format

```
Clib.sinh(floatNum)
```

Table 169 describes the arguments for the Clib Get Hyperbolic Sine method.

Table 169. Arguments for the Clib Get Hyperbolic Sine Method

Argument	Description
floatNum	A floating-point number or a variable that contains a floating-point number. This method calculates the hyperbolic sine of this number.

Clib Get Hyperbolic Tangent Method

The Clib Get Hyperbolic Tangent method calculates and returns the hyperbolic tangent of a floating-point number.

Format

```
Clib.tanh(floatNum)
```

Table 170 describes the arguments for the Clib Get Hyperbolic Tangent method.

Table 170. Arguments for the Clib Get Hyperbolic Tangent Method

Argument	Description
floatNum	A floating-point number or a variable that contains a floating-point number that this method calculates.

Clib Get Integer Method

The Clib Get Integer method calculates and returns the integer part of a decimal number. The effect is identical to that of the Convert Value to Integer method. For more information, see “Convert Value to Integer Method” on page 165.

Format

```
Clib.modf(number, var intVar)
```

Table 171 describes the arguments for the Clib Get Integer method.

Table 171. Arguments for the Clib Get Integer Method

Argument	Description
number	The floating-point number that this method splits.
intVar	Contains the integer part of the number.

Example

The following example passes the same value to the Clib Get Integer method and to the Convert Value to Integer method. The result is the same for each method:

```
function eScript_Click ()
{
   Clib.modf(32.154, var x);
   var y = ToInteger(32.154);
   TheApplication().RaiseErrorText("modf yields " + x +
         ".\nToInteger yields " + y + ".");
}
```

This example produces the following result:

```
modf yields 32
ToInteger yields 32.
```

Clib Get Normalized Mantissa Method

The Clib Get Normalized Mantissa method converts a number into a normalized mantissa in a value in the range of 0.5 through 1.0, and then calculates an integer exponent of 2 so that the number is equivalent to the following value:

mantissa multiplied by 2 ^ exponent

It returns one of the following values:

- A normalized mantissa in the range of 0.5 through 1.0
- 0

A *mantissa* is the decimal part of a natural logarithm.

Format

`Clib.frexp(`*`number, exponent`*`)`

Table 172 describes the arguments for the Clib Get Normalized Mantissa method.

Table 172. Arguments for the Clib Get Normalized Mantissa Method

Argument	Description
number	The number on which this method operates.
exponent	The exponent that this method uses.

Clib Initialize Random Number Generator Method

The Clib Initialize Random Number Generator method initializes a random number generator.

Format

`Clib.srand(`*seed*`)`

Table 173 describes the arguments for the Clib Initialize Random Number Generator method.

Table 173. Arguments for the Clib Initialize Random Number Generator Method

Argument	Description
seed	The number that the random number generator uses as a starting point. If you do not specify the seed argument, then this method uses a random number that is specific to the operating system.

Related Topics

For more information, see the following topics:

- "Get Random Number Method" on page 189
- "Clib Create Random Number Method" on page 265

Clib Date and Time Methods

This topic describes Clib date and time methods. It includes the following topics:

Overview of Clib Date and Time Methods

The Clib time object measures time in the following ways:

- As an integral value of the number of seconds that have occurred since January 1, 1970.
- As a time object that includes properties for the day, month, year, and so on. This time object is distinct from the standard JavaScript date object.

Note the following:

- The time object is for use with the date and time functions in the Clib object.
- You cannot write code that uses a date object property with a time object or a time object property with a date object.
- Although the time object is different than the date object, these objects contain similar data.

Table 174 lists the integer properties for the timeInt argument of the Clib time object.

Table 174. Integer Properties of the timeInt Argument of the Clib Time Object

Value for the timeInt Argument	Integer Property
tm_sec	Second after the minute, from 0.
tm_min	Minutes after the hour, from 0.
tm_hour	Hour of the day, from 0.
tm_mday	Day of the month, from 1.
tm_mon	Month of the year, from 0.
tm_year	Years since 1900, from 0.
tm_wday	Days since Sunday, from 0.
tm_yday	Day of the year, from 0.
tm_isdst	Flag for Daylight Savings Time.

About the Objects That Each Clib Time Method Returns

Table 175 lists the object that each Clib time method returns. *Time* includes a variable in the Time object format, while *timeInt* includes a time value that is an integer.

Table 175. Time Methods and the Objects They Return

Method	Object Returned
Clib Get Date and Time Method	Time
Clib Get Tick Count Method	CPU tick count
Clib Divide Method	timeInt
Clib Get Difference in Seconds Method	timeInt
Clib Convert Integer to GMT Method	timeInt
Clib Convert Integer to Local Time Method	timeInt
Clib Convert Time Object to Integer Method	Time
Clib Get Formatted Date and Time Method	Time
Clib Create Temporary File Name Method	timeInt

Clib Convert Integer to GMT Method

The Clib Convert Integer to GMT method uses the integer value that the Clib Convert Time to Integer method returns and converts it to a time object that includes the current date and time expressed as Greenwich mean time (GMT).

It is recommended that you use the Clib Convert Integer to GMT method only if you cannot use the equivalent standard JavaScript method. Note the following code:

```
var now = Clib.asctime(Clib.gmtime(Clib.time())) + "GMT";
```

This code is exactly equivalent to the following standard JavaScript code:

```
var aDate = new Date;
var now = aDate.toGMTString()
```

Format

`Clib.gmtime(`*`timeInt`*`)`

This method uses the same arguments as the Clib Get Date and Time method. For more information, see Table 178 on page 276.

Example

The following example returns the current GMT date and time:

```
TheApplication().RaiseErrorText(Clib.asctime(Clib.gmtime(Clib.time())));
```

It returns this value as a string that uses the following format:

Day Mon dd hh:mm:ss yyyy:

Related Topics

For more information, see the following topics:

- "Get Day of Month Method" on page 129
- "Convert Date to GMT String Method" on page 127
- "Get Time Method" on page 132
- "Get UTC Day of Month Method" on page 141
- "Clib Divide Method" on page 266

Clib Convert Integer to Local Time Method

The Clib Convert Integer to Local Time method returns the value of the timeInt argument as a time object. It is recommended that you use this method only if you cannot use the equivalent standard JavaScript method. Note the following code:

```
var now = Clib.asctime(Clib.localtime(Clib.time()));
```

This code is exactly equivalent to the following standard JavaScript code:

```
var aDate = new Date;

var now = aDate.toLocaleString()
```

Format

`Clib.localtime(`*`timeInt`*`)`

This method uses the same arguments as the Clib Get Date and Time method. For more information, see Table 178 on page 276.

Related Topics

For more information, see the following topics:

- "Get Day of Month Method" on page 129
- "Get Time Method" on page 132
- "Get UTC Day of Month Method" on page 141

Clib Convert Time to Integer Method

The Clib Return Time in Integers method returns the current time expressed in integers. The time format is not specifically defined except that it includes the current time according to the closest approximation that the operating system can make.

The following code assigns the current local time to the timeInt argument:

```
Clib.time(timeInt) and timeInt = Clib.time()
```

Format

`Clib.time([[var] timeInt])`

Table 176 describes the arguments for the Return Time in Integers method.

Table 176. Arguments for the Return Time in Integers Method

Argument	Description
timeInt	Holds the value that this method returns. You must declare this argument as a variable.

Example

For examples, see the following topics:

- "Clib Divide Method" on page 266
- "Clib Convert Integer to GMT Method" on page 272
- "Clib Convert Integer to Local Time Method" on page 273
- "Clib Get Formatted Date and Time Method" on page 277
- "Clib Get Difference in Seconds Method" on page 279

Related Topics

For more information, see the following topics:

- "Convert Date to Integer Method" on page 126
- "Get Day of Month Method" on page 129

Clib Convert Time Object to Integer Method

The Clib Convert Time Object to Integer method converts a time object to the time format that the Clib Convert Time to Integer method returns. It returns one of the following values:

- If it can convert the value in the Time argument, then it returns the value that the Time argument contains expressed as an integer.
- If it cannot convert the value in the Time argument, then it returns negative 1.

It sets any element of the Time argument that is not defined to 0 before it performs the conversion. This method is the opposite of the Convert Integer to Local Time method that converts a time integer to a time object.

Format

`Clib.mktime(`*`Time`*`)`

Table 177 describes the arguments for the Clib Convert Time Object to Integer method.

Table 177. Arguments for the Clib Convert Time Object to Integer Method

Argument	Description
Time	A time object.

Example

The following example uses the Clib Convert Time Object to Integer method to format a time so that Siebel eScript can use it with the Clib Get Difference in Seconds method:

```
// create time object and set time to midnight:
var midnightObject = Clib.localtime(Clib.time());
midnightObject.tm_hour = 0;
midnightObject.tm_min = 0;
midnightObject.tm_sec = 0;

// use mktime to convert Time object to integer:
var midnight = Clib.mktime(midnightObject);

// difftime can now use this value:
var diff = Clib.difftime(Clib.time(), midnight);
TheApplication().Trace("Seconds since midnight: " + diff);
```

This example produces the following result:

```
COMMENT,Seconds since midnight: 59627
```

For an example that describes the difference between the formats that asctime and mktime use, see "Clib Get Date and Time Method" on page 276.

Related Topics

For more information, see the following topics:

- "Get Day of Month Method" on page 129
- "Get Time Method" on page 132
- "Get UTC Day of Month Method" on page 141

Clib Get Date and Time Method

The Clib Get Date and Time method returns a string that includes the date and time that it extracts from a time object. The string it returns uses the following format:

Day Mon dd hh:mm:ss yyyy

For example, `Wed Aug 10 13:21:56 2005`.

Format

`Clib.asctime(`*`Time`*`)`

Table 178 describes the arguments for the Clib Get Date and Time method.

Table 178. Arguments for the Clib Get Date and Time Method

Argument	Description
Time	A time object.

Example

The following example describes the difference between the asctime and mkdir formats for time:

```
TheApplication().TraceOn("c:\\eScript_trace.txt","allocation","all");

var tm = Clib.localtime(Clib.time());
var tmStr = Clib.asctime(tm);
var tmVal = Clib.mktime(tm);

TheApplication().Trace("Time String : " + tmStr);
TheApplication().Trace("Time Value  : " + tmVal);

TheApplication().TraceOff();
```

This example produces the following result:

```
03/05/04,12:26:30,START,7.5.3 [16157] LANG_INDEPENDENT,SADMIN,6532,6584
03/05/04,12:26:30,COMMENT,"Time String : Fri Mar 05 12:26:30 2004"
03/05/04,12:26:30,COMMENT,Time Value  : 1078489590
03/05/04,12:26:30,STOP
```

Related Topics

For more information, see the following topics:

- “Get Day of Month Method” on page 129
- “Get Time Method” on page 132
- “Get UTC Day of Month Method” on page 141

Clib Get Formatted Date and Time Method

The Clib Get Formatted Date and Time method creates a string that includes the date, time, or the date and time. It returns a formatted string that contains these values.

Format

`Clib.strftime(`*`stringVar, formatString, Time`*`)`

Table 179 describes the arguments for the Clib Get Formatted Date and Time method.

Table 179. Arguments for the Clib Get Formatted Date and Time Method

Argument	Description
stringVar	A variable that holds the time in a string.
formatString	A string that describes how to format the value in the stringVar argument. Conversion characters represent this format. For more information, see "Conversion Characters That the Return Formatted Date and Time Method Uses" on page 277.
Time	A time object that the Clib Convert Integer to Local Time method returns. For more information on the time object, see "Overview of Clib Date and Time Methods" on page 271.

Conversion Characters That the Return Formatted Date and Time Method Uses

Table 180 describes conversion characters that the Return Formatted Date and Time method uses.

Table 180. Conversion Characters That the Return Formatted Date and Time Method Uses

Character	Description	Example
%a	Abbreviated weekday name.	Sun
%A	Full weekday name.	Sunday
%b	Abbreviated month name.	Dec
%B	Full month name.	December
%c	Date and time.	Dec 2 06:55:15 1979
%d	Two digit day of the month.	02
%H	Two digit hour of the 24-hour day.	06
%I	Two digit hour of the 12-hour day.	06
%j	Three digit day of the year from 001.	335
%m	Two digit month of the year from 01.	12
%M	Two digit minute of the hour.	55

Table 180. Conversion Characters That the Return Formatted Date and Time Method Uses

Character	Description	Example
%p	AM or PM.	AM
%S	Two digit seconds of the minute.	15
%U	Two digit week of the year where Sunday is the first day of the week.	48
%w	Day of the week where Sunday is 0.	0
%W	Two digit week of the year where Monday is the first day of the week.	47
%x	The date.	Dec 2 1979
%X	The time.	06:55:15
%y	Two digit year of the century.	79
%Y	The year.	1979
%Z	The name of the time zone, if known.	EST
%%	The percentage symbol.	%

Example 1

The following example displays the full day name and month name of the current day:

```
var TimeBuf;
Clib.strftime(TimeBuf,"Today is %A, and the month is %B",
   Clib.localtime(Clib.time()));
TheApplication().RaiseErrorText(TimeBuf);
```

The display is similar to the following:

```
Today is Friday, and the month is July
```

Example 2

The following example uses various conversion characters to format the value that the Clib Get Formatted Date and Time method returns:

```
TheApplication().TraceOn("c:\\eScript_trace.txt","allocation","all");

var tm, tmStrFmt;
tm = Clib.localtime(Clib.time());

Clib.strftime(tmStrFmt, "%m/%d/%Y",tm);
TheApplication().Trace("Time String Format: " + tmStrFmt);

Clib.strftime(tmStrFmt, "%A %B %d, %Y",tm);
TheApplication().Trace("Time String Format: " + tmStrFmt);

TheApplication().TraceOff();
```

This example produces the following result:

```
03/05/04,12:44:01,START,7.5.3 [16157] LANG_INDEPENDENT,SADMIN,6848,6708
03/05/04,12:44:01,COMMENT,Time String Format: 03/05/2004
03/05/04,12:44:01,COMMENT,"Time String Format: Friday March 05, 2004"
03/05/04,12:44:01,STOP
```

Clib Get Local Date and Time Method

The Clib Get Local Date and Time method returns a string that includes the date and time, adjusted for the local time zone. It is equivalent to the following code:

```
Clib.asctime(Clib.localtime(timeInt));
```

where:

- *timeInt* is the date and time that the Clib Get Date and Time method returns.

Format

```
Clib.ctime(timeInt)
```

Table 181 describes the arguments for the Clib Get Local Date and Time method.

Table 181. Arguments for the Clib Get Local Date and Time Method

Argument	Description
timeInt	The date and time value that this method returns.

Example

The following example returns the current date and time:

```
TheApplication().RaiseErrorText(Clib.ctime(Clib.time()));
```

It returns this date and time in a string that uses the following format:

Day Mon dd hh:mm:ss yyyy

Clib Get Difference in Seconds Method

The Clib Get Difference in Seconds method returns the difference in seconds between two times.

Format

```
Clib.difftime(timeInt1, timeInt0)
```

Table 182 describes the arguments for the Clib Get Difference in Seconds method.

Table 182. Arguments for the Clib Get Difference in Seconds Method

Argument	Description
timeInt0	An integer time value that this method returns.
timeInt1	An integer time value that this method returns.

Example

The following example displays the difference in seconds between two times:

```
function difftime_Click ()
{
   var first = Clib.time();
   var second = Clib.time();
TheApplication().RaiseErrorText("Elapsed time is " +
      Clib.difftime(second, first) + " seconds.");
}
```

Clib Get Tick Count Method

The Clib Get Tick Count method returns the current processor tick count. The count starts at 0 when Siebel CRM starts running and increments the number of times per second according to operating system settings.

Format

```
Clib.clock()
```

Clib Character Classification Methods

This topic describes Clib character classification methods that the Clib object supports. It includes the following topics:

Overview of Clib Character Classification Methods

Siebel eScript does not include character types. For example, a char character is actually a string that is one character in length. Actual usage is similar to the C programming language. For example, the following Clib Is Alphanumeric method works properly:

```
var t = Clib.isalnum('a');

var s = 'a';
var t = Clib.isalnum(s);
```

This code displays the following output:

```
true
true
```

The Clib Is Alphanumeric method in the following example causes errors because the each argument to each statement is a string that contains more than one character:

```
var t = Clib.isalnum('ab');

var s = 'ab';
var t = Clib.isalnum(s);
```

A character classification method returns one of the following values:

- True
- False

Clib Is Alphabetic Method

The Clib Is Alphabetic method returns True if the value you specify in the char argument is one of the following values:

- An alphabetic character from A through Z
- An alphabetic character from a through z

If the value you specify is not one of the these values, then it returns Null.

Format

`Clib.isalpha(`*char*`)`

Table 183 describes the arguments for the Clib Is Alphabetic method.

Table 183. Arguments for the Clib Is Alphabetic Method

Argument	Description
char	A single character or a variable that contains a single character.

Clib Is Alphanumeric Method

The Clib Is Alphanumeric method returns True if the value you specify in the char argument is one of the following values:

- An alphabetic character from A through Z
- An alphabetic character from a through z
- A digit from 0 through 9

If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.isalnum(char)
```

Clib Is ASCII Method

The Clib Is ASCII method returns True if the value you specify in the char argument is an ASCII code from 0 to 127. If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.isascii(char)
```

Clib Is Control Method

The Clib Is Control method returns True if the value you specify in the char argument is a control character that an ASCII code from 0 through 31 represents. If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.iscntrl(char)
```

Clib Is Digit Method

The Clib Is Digit method returns True if the value you specify in the char argument is a decimal digit from 0 through 9. If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.isdigit(char)
```

Clib Is Lowercase Method

The Clib Is Lowercase method returns True if the value you specify in the char argument is a lowercase alphabetic character from a through z. If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.islower(char)
```

Clib Is Printable Method

The Clib Is Printable method returns True if the value that you specify in the char argument is a printable character that you can enter from the keyboard and that an ASCII code 32 through 126 represents. If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.isprint(char)
```

Clib Is Printable Not Space Method

The Clib Is Printable Not Space method returns True if the value you specify in the char argument is a printable character other than the space character that ASCII code 32 represents. If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.isgraph(char)
```

Clib Is Punctuation Mark Method

The Clib Is Punctuation Mark method returns True if the value that you specify in the char argument is a punctuation mark that you can enter from the keyboard. If the value you specify is not one of these values, then it returns Null.

This method returns True if one of the following ASCII codes represents the punctuation mark:

- 33 through 47
- 58 through 63
- 91 through 96
- 123 through 126

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.ispunct(char)
```

Clib Is Space Method

The Clib Is Space method returns True if the value you specify in the char argument is a white space character. If the value you specify is not one of the these values, then it returns Null.

Table 184 describes the items for which the Clib Is Space method returns a value of true.

Table 184. items for Which the Clib Is Space Method Returns a Value of True

Description	ASCII Value
Horizontal tab	9
Newline	10

Table 184. items for Which the Clib Is Space Method Returns a Value of True

Description	ASCII Value
Vertical tab	11
Form feed	12
Carriage return	13
Space character	32

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

For more information, see "Use White Space to Improve Readability" on page 56.

Format

```
Clib.isspace(char)
```

Clib Is Uppercase Method

The Clib Is Uppercase method returns True if the value you specify in the char argument is an uppercase alphabetic character from A through Z. If the value you specify is not one of the these values, then it returns Null.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.isupper(char)
```

Clib Is Hexadecimal Method

The Clib Is Hexadecimal method returns True if the value you specify in the char argument is a hexadecimal character. If the value you specify is not one of the these values, then it returns Null.

A hexadecimal character is one of the following:

- A number from 0 through 9
- An alphabetic character from a through f.
- An alphabetic character from A through F.

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Format

```
Clib.isxdigit(char)
```

Clib Error Methods

This topic describes clib error methods. It includes the following topics:

Clib Clear Error Method

The Clib Clear Error method clears the error status and resets the end-of-file flag for a file that you specify. For usage information, see "Overview of Clib File Input and Output Methods" on page 230.

Format

```
Clib.clearerr(filePointer)
```

Table 185 describes the arguments for the Clib Clear Error method.

Table 185. Arguments for the Clib Clear Error Method

Argument	Description
filePointer	Identifies the file name.

Clib Get Error Number Method

The Clib Get Error Number method determines if an error has occurred in the buffer where Siebel eScript reads a file. It returns one of the following values:

- If no error exists, then it returns the following value:

  0

- If an error exists, then it returns the error number.

Format

```
Clib.ferror(filePointer)
```

The arguments for this method are the same as the arguments for the Clib Clear Error method. For more information, see "Arguments for the Clib Clear Error Method" on page 286.

Related Topics

For more information, see "Clib Error Number Property" on page 288.

Clib Get Error Message Method

The Clib Get Error Message method returns the descriptive error message that is associated with the error number that the error number property identifies. When some methods fail to run properly they store a number in the error number property. This number corresponds to the type of error encountered. The Clib Get Error Message method converts this error number to a descriptive string and returns it.

Format

```
Clib.strerror(ToNumber(Clib.errno)
```

Related Topics

For more information, see "Clib Error Number Property" on page 288.

Clib Save Error Message In String Method

The Clib Save Error Message In String method is identical to the Clib Get Error Message except if you specify the errmsg argument, then the Save Error Message In String method saves the error message in this argument as a string.

Format

```
Clib.perror([errmsg])
```

Table 186 describes the arguments for the Save Error Message In String method.

Table 186. Arguments for the Save Error Message In String Method

Argument	Description
errmsg	An argument that contains the message that describes the error.

Clib Error Number Property

The Clib Error Number property stores an error number if a method fails to run correctly. Many methods in the Clib and Siebel library objects set errno to a nonzero value when an error occurs. Siebel eScript implements errno as a macro to the internal function errno. For more information, see "Siebel Library Methods" on page 201.

Format

```
Clib.errno
```

Usage

To return the error number stored in the Clib.errno property, you use the following ToNumber conversion method:

```
ToNumber(Clib.errno)
```

For more information, see "Convert Value to Number Method" on page 169.

You cannot use Siebel eScript code to modify the errno property. It is available only for read-only access.

You can configure Siebel CRM to reference the error message that is associated with a Clib error number. For more information, see "Clib Get Error Message Method" on page 287.

Other Clib Methods

This topic describes other Clib methods. It includes the following topics:

Clib Convert Character to ASCII Method

The Clib Convert Character to ASCII method clears every bit of the value that the char argument contains except for the seven least significant bits. The result is a seven-bit C representation of the character. It returns this value as a seven-bit ASCII representation.

If the value you specify in the char argument is already a seven-bit ASCII character, then it does not clear any bits and returns the character.

Format

```
Clib.toascii(char)
```

The arguments for this method are the same as the arguments for the Clib Is Alphabetic method. For more information, see Table 183 on page 282.

Example

The following example returns the close parenthesis character:

```
TheApplication().RaiseErrorText(Clib.toascii("©"));
```

Related Topics

For more information, see “Clib Is ASCII Method” on page 282.

Clib Modify Environment Variable Method

The Clib Modify Environment Variable method creates an environment variable, sets the value of an existing environment variable, or removes an environment variable. It returns one of the following values:

- If it is successful, then it returns the following value:

  0

- If it is not successful, then it returns negative 1.

The Clib Modify Environment Variable method does the following:

- Sets the environment variable that the varName argument identifies to the value that the stringValue argument contains.
- Any modification that it makes to an environment variable persists only while the Siebel eScript code and any process that this code calls is running. After this code runs, the environment variable reverts to the value it contained before this method modified this value.
- Automatically removes any environment variable it creates after it finishes.

Format

```
Clib.putenv(varName, stringValue)
```

Table 187 describes the arguments for the Clib Modify Environment Variable method.

Table 187. Arguments for the Clib Modify Environment Variable Method

Argument	Description
varName	The name of an environment variable, enclosed in quotes.
stringValue	The value that this method assigns to the environment variable, enclosed in quotes. If the value in the stringValue argument is null, then this method removes the environment variable that the varName argument identifies.

Example

The following example creates an environment variable and assigns a value to it. To confirm that the variable was created, it then traces the return value:

```
TheApplication().TraceOn("c:\\eScript_trace.txt","allocation","all");
var a = Clib.putenv("TEST","test value");
TheApplication().Trace("TEST        : " + a);
TheApplication().Trace("TEST= " + Clib.getenv("TEST");
TheApplication().TraceOff();
```

This example produces the following result:

```
03/05/04,16:56:28,START,7.5.3 [16157] LANG_INDEPENDENT,SADMIN,3388,7448
03/05/04,16:56:28,COMMENT,TEST        : 0
03/05/04,16:56:28,COMMENT,TEST= test value
03/05/04,16:56:28,STOP
```

Clib Get Environment Variable Method

The Get Environment Variable method returns the value of an environment variable.

Format

`Clib.getenv(`*`varName`*`)`

Table 188 describes the arguments for the Get Environment Variable method.

Table 188. Arguments for the Get Environment Variable Method

Argument	Description
varName	The name of an environment variable, enclosed in quotes.

Example

The following example returns the value of the PATH environment variable:

```
TheApplication().RaiseErrorText("PATH= " + Clib.getenv("PATH"));
```

Clib Send Command Method

The Clib Send Command method sends a command to the command processor for the operating system and opens an operating system window where it runs. After completing the command, it closes this window. It returns the value that the command processor returns. For an alternative that does not open a window, see "Siebel Library Call DLL Method" on page 201.

Format

`Clib.system(`*`commandString`*`)`

Table 189 describes the arguments for the Clib Send Command method.

Table 189. Arguments for the Clib Send Command Method

Argument	Description
commandString	Contains the name of a valid operating system command. This value can include a formatted string followed by variables. For more information, see "Characters That Format Values" on page 231.

Example

The following example displays a directory in a DOS window:

```
Clib.system("dir /p C:\\Backup");
```

Clib Search Array Method

The Clib Search Array method searches an array for a value that you specify. It returns one of the following values:

- If it finds the value you specify in the key argument, then it returns an array variable that matches the value you specify in the key argument.
- If it does not find the value you specify in the key argument, then it returns the following value:

  Null

It only searches through array elements that include a positive index. It ignores array elements that include a negative index.

Format

`Clib.bsearch(`*`key, arrayToSort,`* `[`*`elementCount`*`,]` *`compareFunction`*`)`

Table 190 describes the arguments for the Clib Search Array method.

Table 190. Arguments for the Clib Search Array Method

Argument	Description
key	The value for which this method searches.
arrayToSort	The name of the array that this method searches.
elementCount	The number of array elements that this method searches. If you do not specify the elementCount argument, then it searches the entire array.
compareFunction	A custom function that can affect the sort order. The value for the compareFunction argument must include the following items: ■ The key argument as the first argument ■ A variable from the array as the second argument

Example

The following example uses Clib.qsort and Clib.bsearch to locate a name and related item in a list:

```
(general) (ListCompareFunction)
function ListCompareFunction(Item1, Item2)
{
   return Clib.strcmpi(Item1[0], Item2[0]);
}

(general) (DoListSearch)
function DoListSearch()
   // create array of names and favorite food
   var list =
   {
      {“Brent”, “salad”},
      {"Laura", "cheese" },
      { "Alby", "sugar" },
      { "Jonathan","pad thai" },
      { "Zaza", "grapefruit" },
      { "Jordan", "pizza" }
   };

   // sort the list
   Clib.qsort(list, ListCompareFunction);
   var Key = "brent";
   // search for the name Brent in the list
   var Found = Clib.bsearch(Key, list, ListCompareFunction);
   // display name, or not found
   if ( Found != null )
      TheApplication().RaiseErrorText(Clib.rsprintf
      ("%s's favorite food is %s\n", Found[0][0],Found[0][1]));
   else
      TheApplication().RaiseErrorText("Can not find name in list.");
}
```

Clib Sort Array Method

The Clib Sort Array method sorts elements in an array, starting with index 0, and then continuing to the value that you specify in the elementCount argument minus 1. This method differs from the Sort Array method in standard JavaScript in the following ways:

- The Clib Sort Array method can sort a dynamically created array.
- the Sort Array method in standard JavaScript works only with an array that an Array statement explicitly creates.

Format

`Clib.qsort(`*`array`*`, [`*`elementCount`*`, ]`*`compareFunction`*`)`

Table 191 describes the arguments for the Clib Sort Array method.

Table 191. Arguments for the Clib Sort Array Method

Argument	Description
array	The array that this method sorts.
elementCount	The number of elements in the array, up to 65,536. If you do not specify the elementCount argument, then this method sorts the entire array.
compareFunction	A custom function that can affect the sort order.

Example

The following example prints a list of colors sorted in reverse alphabetical order, ignoring case:

```
// initialize an array of colors
var colors = { "yellow", "Blue", "GREEN", "purple", "RED",
"BLACK", "white", "orange" };
// sort the list using qsort and our ColorSorter routine
Clib.qsort(colors,"ReverseColorSorter");
// display the sorted colors
for ( var i = 0; i <= getArrayLength(colors); i++ )
   Clib.puts(colors[i]);

function ReverseColorSorter(color1, color2)
// do a simple string that is not case-sensitive
// comparison, and reverse the results too
{
   var CompareResult = Clib.stricmp(color1,color2)
   return( _CompareResult );
}
```

This example produces the following output:

```
yellow
white
RED
purple
orange
GREEN
Blue
BLACK
```

Related Topics

For more information, see “Sort Array Method” on page 86.

7 Siebel eScript Quick Reference

This chapter describes summary information for Siebel eScript methods and properties. It includes the following topics:

File and Directory Methods

This topic describes file and directory methods.

File Manipulation Methods

Table 192 describes file control methods.

Table 192. Quick Reference for File Control Methods

Method	Description
Clib Close File Method	Closes an open file.
Clib Create Temporary File Method	Creates a temporary file.
Clib Create Temporary File Name Method	Gets a temporary file name.
Clib Delete File Method	Deletes a file.
Clib Lock File Method	Handles file locking and unlocking.

Table 192. Quick Reference for File Control Methods

Method	Description
Clib Open File Method	Opens a file.
Clib Rename File Method	Renames a file.
Clib Reopen File Method	Reopens a file.

File Manipulation Methods

Table 193 describes file manipulation methods.

Table 193. Quick Reference for File Manipulation Methods

Method	Description
Clib Clear Buffer Method	Writes to disk the data that exists in the buffer, and then clears the buffer.
Clib End of File Method	Determines if the file cursor is at the end of the file.
Clib Get Character Method	Gets a character from the buffer.
Clib Get Characters to Next Line Method	Gets a string that includes characters from the cursor to the next newline character.
Clib Get Cursor Position Method	Gets the current position of the file cursor.
Clib Get Relative Cursor Position Method	Gets the position of the file cursor relative to the beginning of the file.
Clib Move Cursor to Beginning of File Method	Moves the file cursor to the beginning of a file.
Clib Read From File Method	Reads data from a file.
Clib Restore Cursor Position Method	Sets the current file cursor to a position that you specify.
Clib Scan and Convert from Input Device Method	Reads input from an input device and stores the data in arguments.
Clib Set Cursor Position Method	Sets the cursor position in a file.
Clib Unget Method	Pushes a character back to a file.
Clib Write Character Method	Writes a character to a file.
Clib Write Formatted String Method	Writes a formatted string to a file.
Clib Write String to File Method	Writes a string to a file.
Clib Write to File Method	Writes data to a file.

Directory Manipulation Methods

Table 194 describes directory methods.

Table 194. Quick Reference for Disk and Directory Methods

Method	Description
Clib Change Directory Method	Changes directory.
Clib Create Directory Method	Creates a directory.
Clib Get Current Working Directory Method	Gets the current working directory.
Clib Remove Directory Method	Removes a directory.

String Methods

Table 195 describes string and byte array methods.

Table 195. Quick Reference for String and Byte Array Methods

Method	Description
Change String to Lowercase Method	Converts a string to lowercase.
Change String to Uppercase Method	Converts a string to uppercase.
Clib Append String Method	Concatenates a portion of one string to another string.
Clib Compare Strings Method	Performs a comparison between two strings.
Clib Convert String to Lowercase Method	Converts a string to lowercase.
Clib Copy String Method	Copies a part of one string to another string.
Clib Get Formatted String Method	Returns a formatted string.
Clib Get Last Substring Method	Searches a string for the last occurrence of a character.
Clib Get Substring Method	Searches a string for a string.
Clib Search String for Character Method	Searches a string for a character.
Clib Search String for Character Set Method	Searches a string for a set of characters.
Clib Search String for Not Character Set Method	Searches a string for a character that is not in a set of characters.
Clib Write Formatted String Method	Writes formatted output to a string.
Compile Regular Expressions Method	Modifies the pattern and attributes that Siebel CRM uses with the current instance of a regular expression object.

Table 195. Quick Reference for String and Byte Array Methods

Method	Description
Create String From Substring Method	Returns a section of a string.
Create String From Unicode Values Method	Converts Unicode values to a string.
Get Character From String Method	Returns the character that resides at a location in a string.
Get Regular Expression from String Method	Returns an array of strings that match a regular expression.
Get Unicode Character From String Method	Returns the Unicode value of the character that resides at a specific position in a string.
Is Regular Expression in String Method	Indicates if a string includes a regular expression.
Parse String Method	Parses a string and returns an array of strings according to a separator.
Replace String Method	Replaces a string with a string that you define.
Search String for Last Substring Method	Returns the position of the last instance of a substring.
Search StringVar for Regular Expression Method	Returns the position of a regular expression.

Array Methods and Properties

Table 196 describes array methods and properties.

Table 196. Quick Reference for Array Methods

Method or Property	Description
Add Array Elements Method	Appends new elements to the end of an array.
Clib Search Array Method	Searches an array for a value that you specify
Clib Sort Array Method	Sorts elements in an array.
Create Array Elements Method	Creates a string of array elements.
Delete Last Array Element Method	Returns the last element of the current array object, and then removes the element from the array.
Get Array Length Method	Returns the length of a dynamically created array.
Get Largest Array Index Method	Returns a number that includes the largest index of an array, plus 1.
Insert Array Elements Method	Inserts new elements into an array.

Table 196. Quick Reference for Array Methods

Method or Property	Description
Reverse Array Order Method	Reverses the order of elements of an array.
Set Array Length Method	Sets the size of an array.
Sort Array Method	Sorts array elements.

Mathematical Methods and Properties

This topic describes mathematical methods and properties.

Numeric Methods

Table 197 describes numeric methods.

Table 197. Quick Reference for Numeric Methods

Method	Description
Clib Create Random Number Method	Creates a pseudo-random number.
Clib Divide Method	Performs integer division and returns a quotient and remainder.
Clib Get Floating Point Number Method	Calculates a floating-point number given a mantissa and an exponent.
Clib Get Integer Method	Returns the integer part of a decimal number.
Clib Get Normalized Mantissa Method	Breaks a real number into a mantissa and an exponent as a power of 2.
Clib Initialize Random Number Generator Method	Creates an initial value for the random number generator.
Get Absolute Value Method	Returns the absolute value of an integer.
Get Ceiling Method	Returns the smallest integer that is not less than the value that the number argument contains.
Get Exponential Method	Computes the exponential function.
Get Floor Method	Returns the greatest integer that is not greater than the value that the number argument contains.
Get Logarithm Method	Calculates the natural logarithm.
Get Maximum Method	Returns the largest of one or more values.
Get Minimum Method	Returns the smallest of one or more values.
Get Random Number Method	Returns a random real number between 0 and 1.
Get Square Root Method	Calculates the square root.

Table 197. Quick Reference for Numeric Methods

Method	Description
Raise Power Method	Calculates x to the power of y.
Round Number Method	Rounds a value up or down.

Trigonometric Methods

Table 198 describes trigonometric methods.

Table 198. Quick Reference for Trigonometric Methods

Method	Description
Clib Get Hyperbolic Cosine Method	Calculates the hyperbolic cosine.
Clib Get Hyperbolic Sine Method	Calculates the hyperbolic sine.
Clib Get Hyperbolic Tangent Method	Calculates the hyperbolic tangent.
Get Arc Cosine Method	Calculates the arc cosine.
Get Arcsine Method	Calculates the arcsine.
Get Arctangent 2 Method	Calculates the arc tangent of a fraction.
Get Arctangent Method	Calculates the arc tangent.
Get Cosine Method	Calculates the cosine.
Get Sine Method	Calculates the sine.
Get Tangent Method	Calculates the tangent.

Mathematical Properties

Table 199 describes mathematical properties, each of which is a numeric constant.

Table 199. Quick Reference for Mathematical Properties

Property	Description
Base E Property	Returns the value of e, which is the base for natural logarithms.
Logarithm 10 E Property	Returns the value of the base 10 logarithm of e.
Logarithm 2 E Property	Returns the value of the base 2 logarithm of e.
Math Natural Logarithm 10 Property	Returns the value of the natural logarithm of 10.
PI Property	Returns the value of pi.
Square Root 1/2 Property	Returns the value of the square root of ½.
Square Root 2 Property	Returns the value of the square root of 2.

BLOB Methods

Table 200 describes BLOB methods.

Table 200. Quick Reference for BLOB Methods

Method	Description
Get BLOB Data Method	Reads data from a specified position in a BLOB.
Get BLOB Size Method	Determines the size of a BLOB.
Write BLOB Data Method	Writes data to a specified position in a BLOB.

Date and Time Methods

Table 201 describes date and time methods.

Table 201. Quick Reference for Date and Time Methods

Method	Description
Clib Convert Integer to GMT Method	Converts a date and time to GMT.
Clib Convert Integer to Local Time Method	Converts an integer to local time.
Clib Convert Time Object to Integer Method	Converts a time object to an integer.
Clib Get Difference in Seconds Method	Computes the difference between two times.
Clib Get Formatted Date and Time Method	Writes a formatted date and time to a string.
Clib Get Local Date and Time Method	Returns a string that includes the local date and time.
Clib Get Tick Count Method	Returns the current processor tick count.
Convert Date and Time to String Method	Converts a date and time to a string.
Convert Date String to Date Object Method	Converts a date string to a date object.
Convert Date to GMT String Method	Converts a date object to a GMT string.
Convert Date to Integer Method	Converts a date object to an integer.
Convert Integer Date to JavaScript Date Method	Converts an integer date to a JavaScript date.
Convert UTC Date to Readable Date Method	Converts a UTC date to a format that a human can read.
Get Day of Month Method	Returns the day of the month.
Get Day of Week Method	Returns the day of the week.
Get Full Year Method	Returns the year as a four digit number.
Get Hours Method	Returns the hour.
Get Milliseconds Method	Returns the millisecond.

Table 201. Quick Reference for Date and Time Methods

Method	Description
Get Minutes Method	Returns the minute.
Get Month Method	Returns the month.
Get Seconds Method	Returns the second.
Get Time Method	Returns the date and time, in milliseconds, of a date object.
Get Time Zone Offset Method	Returns the difference, in minutes, from GMT.
Get UTC Day of Month Method	Returns the UTC day of the month.
Get UTC Day of Week Method	Returns the UTC day of the week.
Get UTC Full Year Method	Returns the UTC year as a four digit number.
Get UTC Hours Method	Returns the UTC hour.
Get UTC Milliseconds Method	Returns the UTC millisecond.
Get UTC Minutes Method	Returns the UTC minute.
Get UTC Month Method	Returns the UTC month.
Get UTC Seconds Method	Returns the UTC second.
Get Year Method	Returns the year as a two digit number.
Set Date Method	Sets the day of the month.
Set Full Year Method	Sets the year as a four digit number.
Set Hours Method	Sets the hour.
Set Milliseconds Method	Sets the millisecond.
Set Minutes Method	Sets the minute.
Set Month Method	Sets the month.
Set Seconds Method	Sets the second.
Set Time Method	Sets the date and time in a date object, in milliseconds.
Set UTC Date Method	Sets the UTC day of the month.
Set UTC Full Year Method	Sets the UTC year as a four digit number.
Set UTC Hours Method	Sets the UTC hour.
Set UTC Milliseconds Method	Sets the UTC millisecond.
Set UTC Minutes Method	Sets the UTC minute.
Set UTC Month Method	Sets the UTC month.

Table 201. Quick Reference for Date and Time Methods

Method	Description
Set UTC Seconds Method	Sets the UTC second.
Set Year Method	Sets the year as a two digit number.

Buffer Methods and Properties

Table 202 lists buffer methods and properties.

Table 202. Quick Reference for Buffer Methods

Method or Property	Description
Clib Compare Memory Method	Compares the contents of two buffers.
Clib Copy Memory Method	Copies bytes from a source buffer to a destination buffer.
Clib Get Memory Method	Searches a buffer for the first occurrence of a character.
Clib Set Memory Method	Sets the bytes in a buffer to a character that you specify.
Create Buffer Method	Returns a section of a buffer.
Create Buffer Method	Returns a new buffer object that includes the data between two positions.
Cursor Position in Buffer Property	Stores the current position of the buffer cursor.
Data in Buffer Property	A reference to the internal data of a buffer.
Get Buffer Data Method	Returns a string that contains the same data as the buffer.
Get Cursor Position Value From Buffer Method	Returns the value of the current cursor position in a buffer.
Get String From Buffer Method	Returns a string that starts from the current cursor position.
Put String in Buffer Method	Puts a string into a buffer.
Put Value in Buffer Method	Puts a value into a buffer.
Siebel Library Get Pointer Address Method	Gets the address in memory of a buffer variable.
Use Big Endian in Buffer Property	Stores a Boolean flag for bigEndian byte ordering.
Use Unicode in Buffer Property	Stores a Boolean flag that specifies whether to use Unicode strings when calling the Get String from Buffer method or the Put String in Buffer method.
Write Byte to Buffer Method	Provides access to individual bytes in the buffer.

Siebel Library Methods

Table 207 describes methods that can manipulate data at specific memory locations in the Siebel Library.

Table 203. Quick Reference for Siebel Library Methods

Method	Description
Siebel Library Get Pointer Address Method	Gets the address in memory of a buffer variable.
Siebel Library Peek Method	Reads and returns data from a position in memory
Siebel Library Write Data Method	Writes data to a specific position in memory.

Conversion Methods

Table 204 describes conversion methods.

Table 204. Quick Reference for Conversion Methods

Method	Description
Convert Number to Exponential Notation Method	Converts a number to exponential notation.
Convert Number to Fixed Decimal Method	Converts a number to a specific number of decimal places.
Convert Number to Precision Method	Converts a number to a specific number of significant digits.
Convert Special Characters to URL Method	Replaces special characters in a string.
Convert String to Floating-Point Number Method	Converts an alphanumeric string to a floating-point decimal number.
Convert String to Integer Method	Converts an alphanumeric string to an integer number.
Convert Unicode to ASCII Method	converts Unicode characters to equivalent ASCII characters.
Convert Value to Boolean Method	Converts a value to the Boolean data type.
Convert Value to Buffer Method	Converts a value to a buffer.
Convert Value to Integer Method	Converts a value to an integer.
Convert Value to Number Method	Converts a value to a number.
Convert Value to Object Method	Converts a value to an object.
Convert Value to String Method	Converts a value to a string.
Convert Value to String Method	Converts a value to a string.

Table 204. Quick Reference for Conversion Methods

Method	Description
Convert Value to Unsigned Integer 16 Method	Converts a value to an unsigned integer.
Convert Value to Unsigned Integer 32 Method	Converts a value to an unsigned large integer.
Evaluate Expression Method	Returns the value in the expression argument.

Character Classification Methods

Table 205 describes character classification methods.

Table 205. Quick Reference for Character Classification Methods

Method	Description
Clib Is Alphabetic Method	Determines if a character is alphabetic.
Clib Is Alphanumeric Method	Determines if a character is alphanumeric.
Clib Is ASCII Method	Determines if a character is an ASCII character.
Clib Is Control Method	Determines if a character is a control.
Clib Is Digit Method	Determines if character is a decimal digit.
Clib Is Hexadecimal Method	Determines if a character is a hexadecimal-digit character.
Clib Is Lowercase Method	Determines if a letter is a lowercase alphabetic letter.
Clib Is Printable Method	Determines if a character is a printable character.
Clib Is Printable Not Space Method	Determines if character is a printing character except for space.
Clib Is Punctuation Mark Method	Determines if a character is a punctuation character.
Clib Is Space Method	Determines if a character is a white-space character.
Clib Is Uppercase Method	Determines if a character is an uppercase alphabetic character.
Is Finite Method	Determines if a value is finite.
Is NaN Method	Determines if a value is not a number (NaN).

Error Handling Methods

Table 206 describes error handling methods.

Table 206. Quick Reference for Error Handling Methods

Method	Description
Clib Clear Error Method	Clears the error status and resets the end-of-file flag for a file.
Clib Get Error Message Method	Returns the error message associated with an error number.
Clib Get Error Number Method	Returns the error number.
Clib Save Error Message In String Method	Saves an error message in a string.
Throw Statement	Stops running code if an error occurs.

Other Methods

Table 207 describes uncategorized methods.

Table 207. Quick Reference for Other Methods

Method	Description
Clib Convert Character to ASCII Method	Converts a character to ASCII.
Clib Get Environment Variable Method	Returns the value of an environment variable.
Clib Modify Environment Variable Method	Creates or modifies environment variable.
Clib Send Command Method	Causes the operating system to run a command.
Siebel Library Call DLL Method	Calls a procedure from a dynamic link library in Microsoft Windows or a shared object in UNIX.
Undefine Method	Makes a variable undefined.

A Compilation Error Messages

This appendix describes error messages that Siebel eScript creates when Siebel Tools compiles ST eScript code. It includes the following topics:

Formats That This Appendix Uses

This appendix uses the following formats:

- The error prefix is the text that displays for all errors in a group of errors. For example,

  Syntax error at Line *line#* position *character#*:

- The message is the unique part of an error message that applies only to a single error. The message can include text appended after an error prefix, or it can include the entire error message.

Format Error Messages

Table 208 describes error messages that can result from incorrect script format. A format error message starts with the following error prefix: Syntax error at line *line#* position *character#*.

Table 208. Format Error Messages in Siebel eScript

Message	Example	Cause

Expected ':'	**Example 1**
	`function main ()`
	`{`
	`var a = false;`
	`var b = a ? 1, 2;`
	`//expect : after 1`
	`}`
	
	Example 2
	`function main ()`
	`{`
	`var a = {prop1:1, prop2};`
	`//expect : after prop2`
	`}`
	
	Example 3
	`function main ()`
	`{`
	`var a = 1;`
	`var b;`
	`switch (a)`
	`{`
	`case 1`
	`//expect : after 1`
	`b =a;`
	`default`
	`//expect : after default`
	`b = 0;`
	`}`
	`}`
		A colon (:) character is required in the context but you did not provide one. To correct the error, you do the following:
		■ In an expression using the conditional operator, make sure you include a colon between the second and third operands.
		■ In a Switch statement, make sure you include a colon after the value in the Case statement. For example, see example 3.

Table 208. Format Error Messages in Siebel eScript

Message	Example	Cause

Expected ';'	function main ()
	{
	for (i=1; i<10 i++)
	//miss ; after i<10
	{
	...
	}
	}
		A semi-colon (;) character is required in the context but you did not provide one.
		A semi-colon is used to end a statement. Make sure you do the following:
		■ End each statement with a semi-colon.
		■ Include a semi-colon in the For Loop statement.

Expected '('	function main <>
	//expect (after main
	{
	...
	}
		The open parenthesis (( ) and the close parenthesis ( )) do not pair up.

Expected ')'.	Not applicable	The open parenthesis (( ) and the close parenthesis ( )) do not pair up.

Expected ']'.	function main ()
	{
	var a = new Array (10);
	a[10 = 1;
	//expect] after a[10 = 1
	}
		The open square bracket ([ ) and the close square bracket ( ]) do not pair up.

Expected '{'.	function main ()
	var a = new Array (1);
	//expect { before var
		The open curly bracket ({) and the close curly bracket (}) do not pair up.

Expected '}'.	Not applicable	The open curly bracket ({) and the close curly bracket (}) do not pair up.

Expected identifier.	function ()
	// expect an identifier after
	// function */
	{
	...
	}
	
	function main ()
	{
	var;
	//expect an identifier after var
	}
		A name is required in the context but you did not provide one. The name can include one of the following items:
		■ Variable
		■ Property
		■ Array
		■ Function name

Table 208. Format Error Messages in Siebel eScript

Message	Example	Cause

Invalid token.	function main ()
	{
	var a = "\u000G";
	// '\u000G' is an invalid
	// unicode character combination
	}
	
	function main ()
	{
	var a = "\u0G";
	// '\u0G' is an invalid hex
	// character combination
	}
		An invalid Unicode character combination or an invalid hex character combination exists.

Expected while.	function main ()
	{
	do
	{
	...
	}
	//expect while on this line
	}
		The Do While statement is not complete. A while method is required to complete the statement but you did not provide one.

Throw must be followed by an expression on the same line.	try
	{
	var a =
	TheApplication().GetService(“Incorrec
	t name”);
	}
	catch(e)
	{
	throw ;
	//The Throw statement expects an
	expression which is not supplied
	//It must be: throw e;
	}
		The Throw statement must be followed by a name that identifies an exception on the same line, but you did not provide an expression of this type.

Invalid continue statement.	function main ()
	{
	continue;
	// continue is not in a loop
	}
		The Continue statement is not in the body of one of the following items:
		■ Do while
		■ While
		■ For
		■ For in

Table 208. Format Error Messages in Siebel eScript

Message	Example	Cause

Invalid Break statement.	function BreakError()
	{
	break;
	// break is not in a valid
	// loop
	}
		The Break statement is not in the body of one of the following items:
		■ Do while
		■ While
		■ For
		■ For in

Invalid return statement. Return statement cannot be used outside the function body.	function fn ()
	{
	….
	}
	return;
	//Return is outside the function
	//body.
		A Return statement exists outside the body of a function but it must exist in the body of a function.

Invalid left-hand side value.	function main ()
	{
	new Object () = 1;
	// new Object () is not a valid
	// left value
	}
		The left hand side value in an assignment operation must be compatible with the value assigned.
		In the example, the New Object statement is an invalid left-hand value for the equal (=) assignment operator. The valid left-hand value must contain a variable.

Invalid regular expression.	var oRegExp:RegExp;
	oRegExp = /[a-c*/;
	// The regular expression is
	// missing the closing]. It
	// must be [a-c]*.
		The regular expression is invalid. For example, the closing square bracket ( ]) is missing.

Semantic Error Messages

Table 209 describes error messages that can result from semantic errors when Siebel Tools compiles ST eScript code. A semantic error message starts with the following error prefix: Semantic Error around line *line#*. For more information, see "Semantic Warnings" on page 316.

Table 209. Semantic Error Messages in Siebel eScript

Message	Examples	Cause

Argument *argument_label* either type does not correct or is not defined.	function main ()
	{
	fn (new Date(), new Date());
	// type of the second parameter
	// mismatches with function
	// definition and cannot be
	// implicitly converted to
	// 'Number' type
	}
	
	function fn (arg1: chars, arg2: Number)
	{
	TheApplication().RaiseErrorText ("fn");
	}
	main ();
		The argument passed to the function is not of the data type specified in the function definition, or is not defined in the function definition.
		
		In the example, the arg2 parameter must be of type Number, as specified in the function definition, but the function passes the parameter, which is in the following string format:
		
		new date

No such predefined property *property_label* in class *object_type*.	function main ()
	{
	delete "123".prop1;
	// prop1 is not a property of
	// String object. Also, because
	// the String object is
	// constructed here by implicitly
	// converting "123", prop1
	// cannot be created dynamically.
	}
		The property is not defined in the class object.
		
		In this example, you can specify only string object properties. The following property is not a property of the string object:
		
		Prop1

[] operator can only apply to Object, Buffer or Array class.	Not applicable	The script is trying to use the [] operator for a type other than an object, buffer, or array.

Table 209. Semantic Error Messages in Siebel eScript

Message	Examples	Cause

Type mismatch: L: *left_type*; R: *right_type.*	**Example 1**
	`function TypeMismatch()`
	`{`
	`  var BC:BusComp;`
	`  var MyDate:Date = new Date();`
	`  BC =MyDate;`
	`// MyDate is not the same data type`
	`// as strongly typed variable BC`
	`}`
	
	Example 1
	`function fn ()`
	`{`
	`  var a: String;`
	`  a = new Date ();`
	`  //Type mismatch: strongly typed`
	`  //String is assigned a Date.`
	`}`
		A value that belongs to one data type is assigned to a strongly typed variable of another data type. For more information, see "Using Strongly Typed and Typeless Variables" on page 43.

Return type is wrong. Defined return type is *return_type*.	`function fn (): Array`
	`{`
	`  return new Date ();`
	`}`
	
	`fn ();`
		The actual return type is different from the defined return type. Siebel CRM cannot implicitly convert the actual return type to the defined type.

No such label *label* defined.	`switch( switch_variable )`
	`{`
	`  case value1:`
	`  statement_block`
	`  break labl;`
	`// where labl is not a valid label`
	`  .`
	`  .`
	`  .`
	`  [default:`
	`  statement_block;]`
	`}`
		The label referenced in a Break statement or a Goto statement is defined. You must make sure the label name is correct and that the label is defined in the code.
		
		In the example, if the Break statement is to resume at the labl location, then you must define the labl label.

Table 209. Semantic Error Messages in Siebel eScript

Message	Examples	Cause

Continue out of loop.	`function ContinueOut()`
	`{`
	`var i =0`
	`while (i<3)`
	`{`
	`i++;`
	`continue Mylabel;`
	`// Mylabel label is defined`
	`//outside of the while loop.`
	`}`
	`Mylabel:`
	`var a=1;`
	`}`
		A continue command attempts to branch to a label that is not in a loop.

Label redefined.	`function LabelError()`
	`{`
	`Outer:`
	`for (var i = 0; i < 5; i++)`
	`{`
	`var j = 0;`
	`Inner:`
	`while (j!=5)`
	`{`
	`j++;`
	`continue Inner;`
	`Inner: //Label Inner is`
	`//redefined.`
	`var b=1;`
	`}`
	`}`
	`}`
		A label already exists that possesses the same name.

function *function_label* is double defined.	`function fn ()`
	`{`
	`TheApplication().RaiseErrorText`
	`("fn");`
	`}`
	`function fn ()`
	`// second declaration of function`
	`// fn is not allowed`
	`{`
	`TheApplication().RaiseErrorText`
	`("fn again");`
	`}`
		A function already exists that possesses the same name.
		In the example, you must define the function with a name other than the following name:
		fn

Table 209. Semantic Error Messages in Siebel eScript

Message	Examples	Cause

Calling function *function_label* with insufficient number of arguments.	function main ()
	{
	fn ();
	// does not provide enough
	//parameters
	}
	
	function fn (arg1: chars, arg2: chars)
	{
	...
	}
		You did not provide all of the arguments that the function requires.
		
		The number of arguments you include must equal the number specified in the function definition.
		
		In the example, the following function requires two character arguments:
		
		fn

Cannot access property *property_name* on native type.	function main ()
	{
	var a:chars = "123";
	a.m_prop = "123";
	// chars is a primitive, so it
	// has no properties
	}
	main ();
		You cannot write code that assigns a property to a variable that is of a primitive data type, such as char, float, or bool.
		
		In the example, because chars is a primitive data type, you cannot assign it to the following property:
		
		a.m_prop

Table 209. Semantic Error Messages in Siebel eScript

Message	Examples	Cause

Object_name is an invalid object type.	function main ()
	{
	var a: Obj1 = "123";
	// where Obj1 is an invalid object
	// type
	
	}
	main ();
		If you use Siebel eScript that is strongly typed, then you must specify a valid data type in the declaration of the variable.
		
		In the example, the following variable is not a defined object type
		
		obj1
		
		For more information, see "Using Strongly Typed and Typeless Variables" on page 43.

Indiscriminate usage of goto.	function main ()
	{
	var obj = new Object();
	with (obj)
	{
	labl:
	TheApplication().RaiseErrorText
	("in with");
	}
	goto labl;
	}
	main ();
		This script uses a Goto statement to attempt a branch to a With statement block from outside of the With statement block.

Semantic Warnings

A semantic warning notifies you that a script will run but it might produce unexpected results or it might not be efficient. A semantic warning does not display during compilation. To view them in Siebel Tools, you choose Debug, and then Check Syntax.

Table 210 describes semantic warnings in eScript when Siebel Tools compiles ST eScript code. Semantic warnings start with the following prefix: Semantic Warning around line *line#*. For more information, see "Semantic Error Messages" on page 312.

Table 210. Semantic Warnings in Siebel eScript

Message	Example	Cause

Undefined identifier *identifier*. Global object will be used to locate the identifier.	`function main ()`
	`{`
	`obj = new Object();`
	`// obj is created without being`
	`// declared with var.`
	`}`
	`main ();`
		An undeclared variable created in a function is not locally defined. Instead, it is created as a property of the Global object.

Variable *variable* might not be initialized.	`function main ()`
	`{`
	`var a;`
	`TheApplication().RaiseErrorText`
	`(a);`
	`}`
	`main ();`
		The variable declared is not explicitly assigned a value. It is recommended that you initialize a variable to a default value when you declare it. For example:
		`var a="";`
		In the example, you must assign the following variable a value so Siebel CRM can display it in the RaiseErrorText function:
		`a`

Label '*label*' is unused and can be removed.	`function main ()`
	`{`
	`var a = 1;`
	`labl:`
	`// labl is unused`
	`TheApplication().RaiseErrorText`
	`(a);`
	`}`
	`main ();`
		A label is defined in the function but none of the following statements use it. In this situation, you can remove this label:
		■ Continue
		■ Break
		■ Goto

Table 210. Semantic Warnings in Siebel eScript

Message	Example	Cause

Calling function *function_label* with insufficient number of arguments.	`function main ()`
	`{`
	`// It is a warning condition`
	`// instead of an error if the`
	`// missing argument is not`
	`// strongly typed.*/`
	`var c = fn ();`
	`}`
	
	`function fn (a, b)`
	`{`
	`return a+b;`
	`}`
	`main ();`
		You did not provide all of the arguments that the function requires.
		
		The number of arguments you provide must equal the number of arguments specified in the function definition.

Type conversion from *data_type1* to *data_type2* cannot succeed.	`function main ()`
	`{`
	`var n: float = "123";`
	`}`
		When the data type of a variable does not match the value assigned to the variable, the ST eScript engine attempts to convert the data type. This conversion process might not be successful.
		
		In the example, the following string value is assigned to a variable that is of a float data type:
		
		`123`
		
		The ST eScript engine attempts to convert this string to a float data type.

Table 210. Semantic Warnings in Siebel eScript

Message	Example	Cause

No such method *method_name*.	`function main ()`
	`{`
	`fn ();`
	`}`
	`main ();`
		The specified method is not defined or the method name is incorrect.
		
		In the example, you must define fn.

Variable *variable* is double declared.	**Example 1**
	`function fn ()`
	`{`
	`for (var n = 0 ; n < 3 ; n++)`
	`{`
	`...;`
	`}`
	`for (var n = 0 ; n < 3 ; n++)`
	`// n is double declared in //`
	`the scope of fn.`
	`{`
	`...;`
	`}`
	`fn ();`
	
	Example 1
	`function main ()`
	`{`
	`var string1 = "a string";`
	`var string1 = “another string”;`
	`// string1 must not be`
	`redeclared.`
	`}`
	`main ();`
		A local variable is declared more than once.
		
		To avoid this warning for the common case in Example 1, you can do the following:
		
		■ Declare the counter variable outside of the for definition.
		
		■ Use the counter variable without var in the for definition.
		
		For example:
		
		`function fn ()`
		`{`
		`var n;`
		`for (n = 0 ; n < 3 ; n++)`
		`{`
		`...`
		`}`
		`for (n = 0 ; n < 3 ; n++)`
		`{`
		`...`
		
		In Example 2, the multiple declarations result in Siebel eScript assigning each declaration that occurs after the first declaration as a simple assignment but with the unnecessary overhead of declaring a variable. Instead, you can use a simple assignment after the first declaration. For example:
		
		`string1 = “another string”.`

Preprocessing Error Messages

Table 211 describes the preprocessing error message created when Siebel Tools compiles ST eScript code. A preprocessing error message indicates a compatibility issue when Oracle's Siebel Tools compiles ST eScript code. A preprocessing error message starts with the following prefix: PreProcess Error.

Table 211. Preprocessing Error Codes in Oracle's Siebel eScript

Message	Example	Cause
Cannot open include file *file_path*.	`#include "mystuff.js" //where mystuff.js does not exist`	The path to the file in an Include statement is not valid.

The #include directive instructs the preprocessor to treat the contents of a specified file as if these contents exist in the source program at the same location where the directive occurs. You can organize constants and other definitions in include files, and then use #include directives to add these definitions to any source file.

Index

D

E

M

N

O

P

Q

R

S

T