EasyLanguage Syntax Errors

Syntax errors are produced when verifying an EasyLanguage statement that is not understood or expected by the PowerEditor. Following is a list of all syntax errors and their description, listed by error number. Each entry includes the description of the error, probable causes of the error, and examples of the correct and incorrect syntax for the offending statement or instruction (where applicable).

61 "Word not recognized by EasyLanguage."

This error is displayed whenever a word is not recognized by the PowerEditor, for example, if it is not an EasyLanguage reserved word, EasyLanguage function, or a declared user-defined variable, array, or input name.

62 "Invalid number."

The PowerEditor displays this message whenever it finds a typographical error in a number. For example, if a letter is inserted by mistake in a number, the number will be highlighted and this error will be displayed. An example of an invalid number is *100.b4*.

63 "Number out of range."

The PowerEditor displays this error whenever it finds a number that is outside the supported range (a number that is too big). The following statement will produce this error:

```
Value1 =  99999999999999999999;
```

65 "Invalid variable name."

The PowerEditor displays this error whenever it finds an invalid name in a variable declaration statement. Variable names cannot start with a number or any special character other than the underline (_).

For example, this error will be generated when the following statement is verified:

```
Variable: $MyVariable(0);
```

66 “Invalid input name.”

The PowerEditor will display this error whenever it finds an invalid name in an input declaration statement. Input names cannot start with a number or any special character other than the underline (_).

For example, this error will be generated when the following statement is verified:

```
Input: $MyInput(0);
```

70 “Array size cannot exceed 2 billion elements.”

Arrays can have up to 2 billion elements. The number of elements is calculated by multiplying all the dimensions of the array. For example, an array declared using the following statement will have 66 elements:

```
Array: MyArray[10,5](0);
```

This array will have rows 0 through 10 and columns 0 though 5, in other words, 11 rows and 6 columns. The resulting number from multiplying the dimensions of the array can’t exceed 2 billion.

74 “Invalid array name.”

The PowerEditor displays this error whenever it finds an invalid name in an array declaration statement. Array names cannot start with a number or any special character other than the underline (_).

For example, this error will be generated when the following statement is verified:

```
Array: $MyArray[10](0);
```

90 “The first jump command must be a begin: (\\hb,\\pb,\\wb).”

This error is displayed when the PowerEditor finds an end *jump command* without a begin *jump command* in a text string. The end jump commands are:

```
\he
\pe
\we
```

Before these commands, a begin *jump command* must be used.

***Note**: When specifying a file name for the Print() or FileAppend() words, files that start with any of the jump commands will produce this error. So a file name "c:\hello.txt" will produce this error as part of the name \he.*

91 "You cannot nest jump commands within other jump commands."

Jump commands are used in commentary-related text string expressions to highlight words, and create links to the online help. *Jump commands* cannot be nested; that is, there cannot be multiple starting *jump commands* without having matching end *jump commands*.

92 "You must terminate all jump commands with ends (\\he,\\pe,\\we)."

This error is displayed when the PowerEditor finds a begin *jump command* without an end *jump command* in a text string. The begin *jump commands* are:

```
\hb
\pb
\wb
```

After these commands, an end *jump command* must be used.

***Note**: When specifying a file name for the Print() or FileAppend() words, files that start with any of the jump commands will produce this error. So a file name "c:\hello.txt" will produce this error as part of the name \he.*

151 "This word has already been defined."

User-defined words (such as variables, arrays, and inputs) need to have unique names. This error is generated when a user-defined word is defined more than once, as in the following example:

```
Input: vac(10);
Variable: vac (0);
```

154 "=, <>, >, >=, <, <= expected here."

This error is displayed when the PowerEditor evaluates complex true-false expressions and it finds an error within the expression.

```
Condition1 = Condition2 = Close;
```

The intention of this statement was to assign a complex true-false value to the variable *Condition1*, by using *Condition2* and a comparison that involves the *Close*. A corrected version would look like this:

```
Condition1 = Condition2 AND Open = Close;
```

155 " '(' expected here."

The left parenthesis was expected before the highlighted word, for example, if you are using a function that requires parameters, and no parameters are listed.

```
Value1 = Average + 10;
```

In this example, the highlight signifies that a parenthesis was expected before the '+' sign.

156 " ')' expected here."

The right parenthesis was expected after the highlighted word; for example, if you are using a function that requires parameters, you must enclose them in parentheses.

```
Value1 = Average(Close, 10;
```

Here, the highlight signifies that a closing parenthesis was expected before the ';'.

157 "Arithmetic (numeric) expression expected here."

This error is displayed whenever the PowerEditor is expecting a number or a numeric expression and it finds a true-false expression, string value, or any other keyword that does not return a numeric expression. For example, the *Average()* function expects two numeric expressions, so the following:

```
Value1 = Average(Condition1, 10);
```

generates an error since *Condition1* is a true-false expression.

158 "An equal sign '=' expected here."

This error is displayed if the equal sign is omitted when assigning a value to a variable, array, or function (writing an assignment statement).

For example, the following statement will cause an error:

```
Value1 10;
```

and would be corrected by adding an equal sign, as in:

```
Value1 = 10;
```

159 "This word cannot start a statement."

Not all words can be used to start a statement. For example, the data word *Close* cannot be used to start a statement. Usually, reserved words that generate some action are used to start statements such as *Buy*, *Plot1*, or *If-Then*.

160 "Semicolon (;) expected here."

All EasyLanguage statements must end with a semicolon. Whenever the PowerEditor finds a word or expression that can be interpreted as a new line, it will place the cursor before this expression and show this error. For example, the following statements will produce this error:

```
Value1 = Close + Open |
Buy Next Bar at Value1 Stop;
```

Given that the word *Buy* is always used at the beginning of a statement to place a trading order, a semicolon is required after the *Open*.

161 "The word THEN must follow an IF condition."

This error is displayed whenever the word *Then* is omitted from an *If-Then* statement. The word *Then* must always follow the condition of the *If-Then* statement. The correct syntax for an *If-Then* statement is:

```
If Condition1 Then {any operation}
```

162 "STOP, LIMIT, CONTRACTS, SHARES expected here."

This error is displayed by the PowerEditor if it finds a numeric expression following a trading verb without including one of the words listed above. A numeric expression can be used in a trading order to determine the number of shares (or contracts) and/or to specify the price of the stop or limit order. For example:

```
Buy Next Bar at Low - Range;
```

is incorrect because it does not include a trading verb after the price **Range**. To be correct, you could add the word *Stop* or *Limit*, as in:

```
Buy Next Bar at Low - Range Stop;
```

163 "The word TO or DOWNTO was expected here."

This error is displayed whenever writing a *For* loop and the word *to* or *downto* is omitted. The correct syntax for a *For* loop is:

```
For Value1 = 1 To 10 Begin
   {statements}
End;
```

165 "The word BAR or BARS expected here."

This error is displayed whenever referencing to a value of a previous bar where the word *Bar* is omitted. For example, the following statement will cause this error:

```
Value1 = Close of 10 Ago;
```

The correct syntax is:

```
Value1 = Close of 10 Bars Ago;
```

166 "The word AGO expected here."

This error is displayed when the PowerEditor finds a reference to any expression for a number of bars ago without using the phrase *Bars Ago*. For example:

```
Value1 = Close of 10 Bars;
```

produces this error because the word *Ago* is missing. The correct syntax for this expression is:

```
Value1 = Close of 10 Bars Ago;
```

167 " '}' was expected before end of file."

In order to add comments to your EasyLanguage, it is necessary to enclose the commentary text in the curly braces '{' and '}'. An error message is displayed when a left curly brace is found without a matching right curly brace.

```
{ this was written by Trader Joe
If Close > Highest(High, 10)[1] Then
   Buy Next Bar at Market; |
```

Above, the right curly brace was omitted somewhere before the vertical cursor. In this example, a right curly brace should have been placed after the word 'Joe'.

168 " '[' was expected here."

When declaring, assigning, or referencing array values you are required to use the squared braces to specify the array element(s). This error is displayed if the left squared brace is not used when working with an array.

```
Array: MyArray(10);
```

For example, here the highlight shows that a squared brace, corresponding to the declared number of array element, is expected before the parenthesis.

169 "']' was expected here."

When working with bar offsets or arrays, the bar or array index must be enclosed in squared braces. This message is displayed if the right squared brace is missing.

```
Value1 = Close[10 * 1.05;
```

In this example, the highlight indicates that a squared bracket should be placed somewhere before the semicolon. Note that since the PowerEditor is expecting a numeric value in the squared braces, it places the highlight after the last character in a numeric expression. However, in this case, the right bracket was probably intended to be placed after the number 10.

170 "Assignment to a function not allowed."

This error is displayed when you attempt to assign a value to a function. By definition, a function is an EasyLanguage procedure that returns a value, so it is not possible to assign a different value to a function (except when returning a value from within a function).

```
Average = 100.1245;
```

In this example, the highlighted function name indicates that you cannot assign it a value.

171 "A value was never assigned to user function."

By definition, a function is a set of statements that return a value. This error will be displayed when editing or creating a function and the PowerEditor finds that no value has been assigned to the function. A statement similar to the following must be included in every function:

```
MyFunction = Value;
```

where *MyFunction* is the name of the function and *Value* is the expression to be returned when the function is referenced.

172 "Either NUMERIC, TRUEFALSE, STRING, NUMERICSIMPLE, NUMERICSERIES, TRUEFALSESIMPLE, TRUEFALSESERIES, STRINGSIMPLE, or STRINGSERIES expected."

When declaring the inputs in a function it is necessary to specify the type of each input. This error is generated when any word or value, other than a valid input type, is used when declaring function inputs.

174 "Function not verified."

In order for an analysis technique to verify, all functions used by the analysis technique must be verified as well. This error is displayed if there is a function that is not verified and you attempt to verify the analysis technique.

In order to solve this, open the function and verify it, or run "Verify All" from the PowerEditor menu.

175 " ',' or ')' expected here."

This error is displayed when listing a number of elements in parentheses and a semicolon is read before the list is finished.

```
Value1 = Average(Close, 10;
```

In this case, the highlight indicates that either more parameters (separated by a comma) or a right parenthesis were expected before the semicolon.

176 "More inputs expected here."

This error is displayed whenever referencing a function or an included strategy without specifying enough inputs. For example:

```
Value1 = Average(Close);
```

displays an error because only one input is specified while the *Average* function requires two inputs: (1) the price to be averaged and (2) the number of bars.

177 "Too many inputs supplied."

The PowerEditor displays this error when too many inputs are supplied for a function. For example, the *Average* function requires only two inputs, so the following statement will produce this error:

```
Value1 = Average(Close, 10, 5);
```

The correct syntax would be

```
Value1 = Average(Close, 10);
```

180 “The word #END was expected before end of file.”

The compiler directive #END must be used to indicate the end of a group of statements included in the alert or commentary-only section of an analysis technique. The alert and commentary compiler directives will allow certain instructions to be executed only when the alert or commentary is enabled.

181 “There can only be 10 dimensions in an array.”

Arrays can have up to 10 dimensions. The correct syntax for creating a multidimensional array is:

```
Array: MyArray[10,10,10](0);
```

where this statement creates a three-dimensional array of 11 × 11 × 11.

183 “More than 100 errors. Verify termination.”

When the PowerEditor is verifying a document for correctness, it will continue to evaluate expressions until it finds 100 errors. These errors will be found in the error log once the verification process is finished. If the PowerEditor finds more than 100 errors, it will stop the process and will display this message.

185 “Either HIGHER or LOWER expected here.”

When specifying the execution instructions for an order in a strategy, it is possible to use the words *or Higher* and *or Lower* as synonyms to stop and limit. This error occurs when the word *or* is found in an order without the words *Higher* or *Lower*. The following is the proper syntax for this statement:

```
Buy Next Bar at Low - Range or Lower;
```

186 “Input name too long.”

Input names in any PowerEditor analysis technique can be up to 20 characters long. This error is displayed by the PowerEditor whenever an input has a name that has more than 20 characters.

187 “Variable name too long.”

Variable names can have up to 20 characters. This error is displayed whenever a variable is declared with a name that contains more than 20 characters.

188 "The word BEGIN expected here."

This error is generated whenever the PowerEditor is expecting a block statement. For example, all loops require *Begin* and *End* block statements, so writing the following will generate this error:

```
For Value1 = 1 To 10
    Value10 = Value10 + Volume[Value1];
```

The correct syntax is:

```
For Value1 = 1 To 10 Begin
    Value10 = Value10 + Volume[Value1];
End;
```

189 "This word not allowed in a strategy."

The word highlighted by the PowerEditor is not allowed in a Strategy.

190 "This word not allowed in a function."

The word highlighted by the PowerEditor is not allowed in a function. Words like *Plot1*, *Buy*, *SellShort*, etc., are not allowed in functions.

191 "This word not allowed in a study."

The word highlighted by the PowerEditor is not allowed in a study. Words like *Plot1*, *Buy*, *SellShort*, etc., are not allowed in studies.

192 "This word not allowed in an ActivityBar."

The word highlighted by the PowerEditor is not allowed in an ActivityBar study. Words like *Plot1*, *Buy*, *SellShort*, etc., are not allowed in ActivityBar studies.

193 "Comma (,) expected here."

Commas are used to separate elements in a list, for example, when declaring multiple inputs or variables, or when listing the parameters of a function.

This error will be generated whenever the PowerEditor finds two words that seem to be part of the list but are not separated by a comma. For example, in the following:

```
Inputs: Price(Close)| Length(10);
```

the comma after the first input is missing. The PowerEditor places the vertical cursor at the location where it was expecting a comma.

195 "Matching quote is missing."

All text string expressions need to be within double quotes. This error will be displayed whenever there are not matching quotes around a text string expression. For example, the following statement will produce this error:

```
Variable: Txt(" ");
Txt = "This is an example;
```

because there is a missing quote to the right of the text expression. The correct syntax for this expression is:

```
Variable: Txt(" ");
Txt = "This is an example";
```

197 "Strategy not verified."

In order for a trading strategy to verify, any strategies referenced by the trading strategy through the use of the *IncludeStrategy* reserved word must be verified as well. This error is displayed if you attempt to verify a trading strategy that references an unverified strategy.

In order to solve this, open the referenced strategy and verify it, or run "Verify All" from the PowerEditor menu.

200 "Error found in function."

This error is displayed whenever verifying an analysis technique that refers to an unverified function. The only solution is to open the function, verify the function, and then return to the analysis technique.

201 "User function cannot refer to current cell of itself."

A simple function cannot refer to the same value of a function within its calculations. However, if defined as a series function, it can refer to a previous value of itself. For example, the following simple function gives an error:

```
MyFunction = MyFunction + Volume;
```

because the calculation refers to the current value of the function. By setting the function **Parameter** to "Series," the following becomes a valid expression that uses a function's previous value to accumulate the volume of the chart:

```
MyFunction = MyFunction[1] + Volume;
```

204 "Orders cannot be inside a loop."

EasyLanguage does not allow trading orders to be placed inside a *For* or *While* loop. If the intention of placing an order inside a loop is to increase the number of shares or contracts that the strategy will handle, this can still be done by placing the calculation of the number of shares or contracts inside a loop and then using the resulting value in the order instruction after the loop is finished. For example,

```
While Condition1 Begin
   Value1 = <calculation of number of shares>;
End;
Buy Value1 Shares Next Bar at Market;
```

205 "Statement does not return a value."

This error is displayed when attempting to return a value from statements not designed to return a value, such as those that set or change a value. For example:

```
Value1 = AB_SetZone (High, Low, RightSide);
```

To correct this error, do not assign the expression to a variable:

```
AB_SetZone(High, Low, RightSide);
```

208 "CONTRACTS, SHARES expected here."

When writing an EasyLanguage statement to place an order, it is possible to specify how many contracts or shares the strategy should use to open (or exit) the position. This error will be generated by the PowerEditor whenever it finds a numeric expression after the trading verb that is not followed by the words *Stop*, *Limit*, *or Higher*, or *or Lower*. For example:

```
Buy 100;
```

generates an error because it is not clear if '100' is a part of the instructions to specify the number of shares or the execution instruction (the price at which the order should be placed). A correct statement might read:

```
Buy 100 Shares;
```

209 "Strategy name expected within quotes."

When specifying the name of an order, it must be enclosed within parentheses and double quotes. This error is displayed if the name is missing or not correctly provided. For example, the following statement will cause this error:

```
Sell From Entry () Next Bar at Market;
```

211 "Strategy cannot call itself."

A strategy cannot reference itself when using the *IncludeStrategy* reserved word.

213 "Error found in strategy."

This error is displayed whenever verifying a strategy that contains the *IncludeStrategy* reserved word that references a strategy that is not verified. The only solution is to open the unverified strategy, verify it, and then return to the original strategy.

214 "Colon (:) expected here."

EasyLanguage expects a colon to be used when declaring certain elements of the language like inputs, variables, arrays, and DLLs. In order to declare a new input, the word 'Input' should be followed by a colon, and then the list of input names. This error will be displayed whenever the colon is missing from this expression, for example:

```
Input MyValue(10);
```

Since there is no colon after the word 'Input', the word *MyValue* is highlighted and this error message is displayed. To correct the error, simply add a colon after 'Input':

```
Input: MyValue(10);
```

215 "Cannot use next bar's price and close order in the same strategy."

EasyLanguage does not support using information from the next bar (the *Date*, *Time*, or *Open*) and placing an order at the close of the current bar in the same strategy. If the instructions are not related, they should be written as different strategies and merged using TradeStation StrategyBuilder.

The following produces an error because it includes a reference to the *Open of Next Bar* with a *Close* order for *This Bar* (current bar):

```
If Open of Next Bar > Price Then Buy This Bar on Close;
```

217 "Function circular reference found."

A circular reference is defined as two formulas that refer to each other in their respective calculations. This type of formula cannot be solved by EasyLanguage, so whenever a circular reference is found this error is displayed.

For example, a circular reference can happen if you have a function *A*, which is defined as the value of the current bar of a function *B* plus 1, and the definition of the function *B* is the value of the current bar of *A* plus 1. In order to calculate the value of function *A*, the value of *B* is needed, but in order to calculate *B*, the value of *A* is needed. Therefore, it is not possible to obtain the values of these functions and this error occurs.

220 “Cannot anchor a global exit.”

The price date of the bar where an entry order was placed can be accessed from an exit by using *At$*. This is allowed only when the entry order has a label and if the exit is tied to the entry. An error will be generated if the entry is not labeled or if the exit does not specify what entry it is attempting to close. For example, the following exit will cause this error:

```
If Condition1 Then
   Buy ("MyEntry") This Bar on Close;
Sell At$ Low - 1 Stop;
```

since the exit does not specify the name of a matching entry. The correct syntax is:

```
If Condition1 Then
   Buy ("MyEntry") This Bar on Close;
Sell From Entry ("MyEntry") At$ Low - 1 Stop;
```

223 “A simple function cannot call itself.”

Historical values of simple functions are not available to EasyLanguage, so referring to previous values of itself in its calculations is not allowed. If this is necessary, change the function to a series.

```
MyFunction = MyFunction [1] + Volume;
```

For example, if *MyFunction* is a simple function, the above reference to the value of *MyFunction* of one bar ago is not allowed in this calculation.

224 “Strategy name already used.”

The PowerEditor does not allow the reuse of a name in two different orders. It is mandatory that all orders have a different name. The following *SellShort* statement produces this error:

```
If Condition1 Then
   Buy ("MyStrategy") Next Bar at Market;
If Condition2 Then
   SellShort ("MyStrategy") Next Bar at Market;
```

because both orders cannot have the same name.

226 "Next bar's prices can only be used in a strategy."

The *Open*, *Date*, and *Time* of the next bar can be referenced only from a strategy; no other analysis has access to this information.

227 "Default expected here."

When declaring an input in any analysis technique, you need to enclose the default value in parentheses. This error will be shown whenever there is no default value specified (the parentheses are empty). For example, the following is the correct syntax for declaring an input with the default value of 15:

```
Input: MyInput(15);
```

229 "Invalid initial value."

An initial value needs to be specified when declaring a variable or array. This initial value needs to be enclosed by parentheses and is used to (1) determine the type of the variable or array (numeric, true-false, or text string), and (2) assign the initial value of the variable or array on the first bar.

The correct syntax when declaring a variable is:

```
Variable: MyVariable(10);
```

where the initial value assigned to this variable is *10*, which is a numeric value.

230 "Initial value expected here."

An initial value needs to be specified when declaring a variable or array. This initial value needs to be enclosed by parentheses and is used to (1) determine the type of the variable or array (numeric, true-false, or text string), and (2) assign the initial value of the variable or array on the first bar.

The correct syntax when declaring a variable is:

```
Variable: MyVariable(10);
```

where the initial value assigned to this variable is *10*, which is a numeric value.

231 "Function has no inputs. Parenthesis not needed."

This error is shown by the PowerEditor when parentheses are used for a function that has no inputs. For example, the EasyLanguage function *Range* has no inputs, so the following statement:

```
Value1 = Range(10);
```

displays the error message and highlights the first parenthesis before the parameter.

232 “Matching left comment brace ‘{’ is missing.”

The PowerEditor displays this error whenever it finds a right comment brace “}” without a matching left comment brace. In order to fix this, find the beginning of the comment text and place a left comment brace before it. If there is no comment in your analysis technique, then remove the right comment brace.

233 “Extra right parenthesis.”

When writing any type of expression or statement that requires parentheses, it is necessary to have matching left and right parentheses. This error is displayed if there are extra right parentheses in the expression being evaluated. For example:

```
Value1 = (Close + Open))/2
```

234 “END found without matching BEGIN.”

This error is displayed whenever a block statement does not contain a matching *End* for every *Begin*.

237 “Position Information function not allowed in a study.”

Strategy position information words can be used only in strategies and functions. This error will be generated if any one of these words are found in anything other than a strategy or function.

238 “Performance Information function not allowed in a study.”

Strategy performance information words can be used only in strategies and functions. This error will be generated if any one of these words is found in anything other than a strategy or function.

239 “Array name too long.”

Array names can have up to 20 characters. An error message will be displayed if the array name used in the declaration statement has more than 20 characters.

240 “This strategy name does not exist.”

This error is displayed whenever tying an exit to a nonexistent entry name. For example, the following strategy produces this error:

```
Buy ("Break") Next Bar at Highest(High, 10) Stop;
Sell From Entry ("BreakOut") Next Bar at Low Stop;
```

because the exit incorrectly refers to an entry labeled “BreakOut,” which does not exist in this strategy. Changing the entry name to “Break” will correct this error.

241 “Cannot exit from an exit strategy.”

This error is displayed when an exit strategy is mistakenly tied to another exit strategy. Exit strategies can be tied to an entry only through the use of the instruction *from Entry (“entry name”)*. For example, the following statements will generate this error:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;

If Condition2 Then
   Sell ("MyExit") This Bar at Close;

Sell from Entry ("MyExit") Next Bar at Lowest(Low,10) Stop;
```

Instead, the following statements are correct:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;

If Condition2 Then
   Sell ("MyExit") This Bar at Close;

Sell From Entry ("MyEntry") Next Bar at Lowest(Low,10) Stop;
```

242 “Cannot BuyToCover from a buy strategy.”

This error will be displayed when a short exit strategy is mistakenly tied to a long entry strategy. Short exit strategies can be tied only to a short entry through the use of the instruction *from Entry (“entry name”)*. For example, the following statements will generate this error:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;
BuyToCover From Entry ("MyEntry") Next Bar at Lowest(Low,10)
Stop;
```

In this case, the error can be corrected by using the appropriate exit instruction, *Sell*:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;
Sell From Entry ("MyEntry") Next Bar at Lowest(Low,10) Stop;
```

243 "Cannot Sell from a SellShort strategy."

This error will be displayed when a long exit strategy is mistakenly tied to a short entry strategy. Long exit strategies can be tied only to a long entry through the use of the instruction *from Entry ("entry name")*. For example, the following statements will generate this error:

```
If Condition1 Then
   SellShort ("MyEntry") This Bar at Close;

Sell from Entry ("MyEntry") Next Bar at Low Stop;
```

In this case, the error can be corrected by using the appropriate exit instruction, *BuyToCover*:

```
If Condition1 Then
   SellShort ("MyEntry") This Bar at Close;

BuyToCover from Entry ("MyEntry") Next Bar at Low Stop;
```

244 "At$ cannot be used after the word TOTAL."

EasyLanguage does not allow the reserved word *Total* to be tied to reference information from the bar of entry by using the *AT$* instruction. For example, the following statement will generate this error:

```
Sell 20 Shares Total From Entry ("MyEntry") At$ Low Stop;
```

247 "References to previous values are not allowed in simple functions."

Prior values of simple functions, simple variables, or simple expressions cannot be referenced from within simple functions. If this is necessary for the calculation of a function, then the function must be set as series, not simple. This incorrect example:

```
MyFunction = MyFunction[1] + Close;
```

creates an error if *MyFunction* is a simple function with a reference to previous values of itself. Setting the function **Properties** to "Series" will correct this error.

250 "Cannot reference a previous value of a simple input."

Historical values of simple inputs in functions are not stored by EasyLanguage, so referring to previous values of them is not allowed. For example, in the following:

```
Input: MyVal(NumericSimple);
MyFunction = MyVal[5];
```

the value *MyVal[5]* is not allowed in this function since it includes a reference to the value of the input of five bars ago but is declared as a *NumericSimple* input. If the reference to a previous value is necessary, change the input type to series.

251 "Variables and arrays not allowed here."

In previous product versions this error is displayed when attempting to pass variables or arrays to series functions.

```
Value2 = Average(Close, Value1);
```

253 "Cannot reference a previous value of this input."

Historical values of simple inputs in functions are not stored by EasyLanguage, so referring to previous values of them is not allowed. For example, in the following:

```
Input: MyVal(NumericSimple);
MyFunction = MyVal[5];
```

the value *MyVal[5]* is not allowed in this function since it includes a reference to the value of the input of five bars ago but was declared as a *NumericSimple* input. If the reference to a previous value is necessary, change the input type to series.

258 "Variables, arrays and inputs not allowed here."

This error is displayed when a variable, array, or input is used as the initializer for an input value, such as:

```
Vars: MyVar(3);
Input: MyInput(MyVar);
```

259 "This number is too big."

The PowerEditor displays this error whenever it finds a number that is outside the supported range (a number that is too big). The following statement produces this error:

```
Value1 = 99999999999999999999;
```

260 " 'Next Bar' can only be applied to 'OPEN', 'DATE' and 'TIME'."

The only prices available from the next bar that can be read from a strategy are *Open*, *Date*, and *Time.* All other prices from the next bar can't be accessed.

261 "The word 'BAR' expected here."

This error is shown whenever writing an order in a strategy where the word *Bar* is left out of the expression. For example, the following:

```
Buy Next on the Close;
```

generates an error because *Bar* is missing. The correct syntax is:

```
Buy Next Bar on the Close;
```

262 "At market order can only be placed for the next bar."

All analysis techniques are read and executed at the end of each bar. Because of this, market orders can be placed only for the next bar. An error will be generated whenever a market order is placed to be filled on this bar, such as:

```
Buy This Bar at Market;
```

263 "Stop and limit orders can only be placed for the next bar."

This error is displayed when trying to write a stop or limit order for the current bar. For example:

```
Buy This Bar at Low - Range Limit;
```

is not correct because a *Limit* order cannot be placed on *This Bar*. To be correct, the *Limit* order must be on the *Next Bar*:

```
Buy Next Bar at Low - Range Limit;
```

264 "On close order must be placed for this bar."

Given that all instructions are read at the close of each bar, the only types of orders that can be placed on the current bar are at the close. Whenever *This Bar* is included as part of an order it may refer only to the *at Close* price. The correct syntax for *This Bar* orders is:

```
Buy This Bar at Close;
```

265 "Cannot mix next bar prices with data streams other than data1."

EasyLanguage prohibits the reference of secondary data streams in the same strategy where references to the *Date*, *Time*, or *Open* of the next price are also made. If the references to a secondary data stream and the next bar prices are not directly related, it is recommended that you write two strategies, one that uses next bar prices and a second that references other data streams.

For example, the following statements included in one strategy are not allowed because they reference two different data streams (*Data1* by default is the first and *Data2* in the second):

```
If Open Next Bar > High Then
   SellShort Next Bar at Open Next Bar + Range Limit;

If Average(Close, 4) of Data2 < Average(Close, 7) of Data2 Then
   BuyToCover Next Bar at Close;
```

Instead, writing two different strategies, one containing the first IF-THEN statement and another containing the second IF-THEN statement is necessary. Later these strategies can both be included as part of the same Trading Strategy by adding them to the same Chart Analysis window.

266 "Library name within double quotes expected here."

The PowerEditor displays this error when defining an external DLL function and the name of the DLL is missing or incorrect. The first element of the list of parameters in the *DefineDLLFunc* statement should be the name of the DLL library within double quotes. The following statement will generate this error:

```
DefineDLLFunc: int, "MyFunc", int;
```

The correct syntax for this statement is:

```
DefineDLLFunc: "MyDLL", int, "MyFunc", int;
```

267 "DLL function name within double quotes expected here."

When defining a function from a DLL, the name of the DLL must be enclosed in double quotes. For example, the following is a proper example of such a function definition because it includes the function name "user.dll" followed by DLL's return type and parameters:

```
DefineDLLFunc: "user.dll", int, "beep";
```

274 "Return type of this DLL function must be specified."

When declaring a DLL function, the return type of the function must be the second parameter listed. Following is a correct DLL function declaration statement with the DLL's type **int** following the DLL name:

```
DefineDLLFunc: "MyDLL.DLL", int, "MyFunction", int;
```

276 "DLL name cannot be longer than 60 characters."

The name of the DLL used to define any function through the *DefineDLLFunc* statement cannot exceed 60 characters.

277 "DLL function name cannot be longer than 65 characters."

The name of a function defined using the *DefineDLLFunc* statement cannot exceed 65 characters.

278 "A variable expected here."

Whenever the PowerEditor expects a variable and finds another reserved or user-defined word, it will highlight the unexpected word and give this message. An example is when a function is expecting a variable as one of the parameters (because it is expecting to receive the variable by reference).

279 "An array expected here."

Functions can now receive arrays as parameters. If a function is expecting an array and instead the PowerEditor finds a variable, input, or other reserved word (different from an array), it will display this error. In the following example the function *Average_a()* calculates the average of a particular array, so the following will generate the syntax error:

```
Variable: MyVar(0);
Value1 = Average_a(MyVar, 10);
```

To correct this problem, you need to declare *MyVar* as an array instead of an integer. It should be written:

```
Array: MyArray[20](0);
Value1 = Average_a(MyArray, 10);
```

280 “TrueFalse expression expected here.”

This error is displayed when the PowerEditor expects a true-false expression and finds a numeric or text string expression instead. For example:

```
Condition1 = High ;
```

281 “Mixing data types (NUMERIC, TrueFalse, String) not allowed.”

This error appears when incompatible data types are combined in a single expression. In this example:

```
Value1 = 100 + "12" ;
```

the text string “12” cannot be directly combined with a numeric value. To resolve such a problem, use the appropriate EasyLanguage reserved word to convert the data to a compatible type.

For example, use the function *StrToNum* to convert the text string to a numeric value:

```
Value1 = 100 + StrToNum("12") ;
```

283 “Strategy has no inputs. Comma not needed.”

When including a strategy through the *IncludeStrategy* keyword, the list of the inputs must be supplied and each input must be separated by a comma. This error is displayed if the strategy has no inputs, and an input is mistakenly included in the statement.

Following is the correct syntax of an *IncludeStrategy* statement of a strategy with no inputs:

```
IncludeStrategy: "My Trailing LX";
```

284 “There is no such strategy.”

This error is displayed by the PowerEditor whenever the strategy name referenced by an *IncludeStrategy* statement does not exist in the strategy library.

285 “Strategy circular reference found.”

A circular reference is defined as two formulas that refer to each other’s current bar value in their respective calculations. This type of formula cannot be solved by EasyLanguage, so whenever a circular reference is found this error is displayed.

286 “Cannot divide by zero.”

This error will be displayed when dividing any numerical expression by the literal number 0. So when the following is written:

```
Value1 = Close / 0;
```

the PowerEditor will generate a syntax error because dividing by zero is a mathematical indetermination and cannot be solved.

287 “File name expected here.”

This error is displayed when using the *Print* statement to send information to the printer, and an invalid file name is used for the file name. The file name should be specified as text between double quotes. Note that a text string expression will not be accepted as a file name in the *Print* statement. For example, the PowerEditor will display this error when evaluating the following statement:

```
Print(File(Value1), Date, Time, Close);
```

The file name needs to be text included in double quotes, for example:

```
Print(File("c:\tradestation\test.txt"), Date, Time, Close);
```

288 “A file or directory name must be <260 characters and may not contain “/ : ∗ ? < > |".”

Certain instructions like the *Print()* and *FileAppend()* statements require a file name. The file name needs to be less than 260 characters long and cannot have any of the characters listed in the error label. For example, this error will be displayed when writing:

```
Print(File("c:\data?.txt", Date, Time, Close);
```

since the ‘?’ character is not a valid character and cannot be used as part of a file name.

291 “The word ‘OVER’ or ‘UNDER’ expected here.”

This error is displayed whenever using the word *Cross* without *Over* or *Under* when writing a true-false expression. For example, the following expression will produce this error:

```
Condition1 = Close Crosses Open ;
```

The correct syntax would be:

```
Condition1 = Close Crosses Over Open;
```

292 "Two constants cannot cross over each other."

The PowerEditor displays this error whenever using the logical operators *Crosses Over* or *Crosses Under* compares two constants. Since they are constants, they will never cross each other and the statement will display an error, as in:

```
Condition1 = 10 Crosses Over 15;
```

293 "This plot has been defined using a different name."

The value of a plot can be assigned more than once within an analysis technique but it must always be referenced using the same name (or the name can be left out). For example, the following statement will cause this error:

```
Plot1( Volume, "Vol" );

If Volume > 1000000 Then
   Plot1(Volume, "V", Red);
```

because the plot has been assigned a second name "V". The correct way of writing this statement is:

```
Plot1( Volume, "Vol" );

If Volume > 1000000 Then
   Plot1(Volume, "Vol", Red);
```

295 "This plot name has never been defined."

This error is displayed whenever referencing a *Plot* with a different name than it was defined with, or a plot that doesn't exist. For example, the following statements will cause this error:

```
Plot1(High, "H");
Value1 = Plot1 + Plot2 ;
```

since Plot2 has not been defined. The PowerEditor highlights the second instance of the *Plot* command to indicate where the error occurred.

296 "This plot has never been assigned a value."

This error is generated when referring to the value of a plot that has not been previously defined in the analysis technique. For example, the following statements will produce this error:

```
Plot1( Average(Close,10) );
If Plot1 Crosses Over Plot2 Then
   Alert;
```

because Plot2 has not been defined.

297 "Server field name too long; cannot be more than 30 characters."

Server Quote fields can be up to 30 characters long. This error will be generated whenever a server field with a name that has more than 30 characters is used.

298 "Strategy Information (for plots) function not allowed in a strategy."

None of the "Strategy Information for plots" words can be used within a strategy. These words are designed to be used in other analysis techniques to refer to overall performance of the strategy. However, there are strategy-specific words that can be used from the strategy to refer to these figures.

These words are:

```
I_AvgEntryPrice
I_ClosedEquity
I_CurrentContracts
I_MarketPosition
I_OpenEquity
```

299 "Strategy Information function not allowed in a study."

Strategy information words (other than the strategy information for plots) can be used only in trading strategies and functions. These words, which are listed in the EasyLanguage Dictionary under the categories *Strategy Performance* and *Strategy Position*, can be used only when writing trading strategies and functions.

300 "This plot has been defined with a different type."

The value of a plot can be assigned more than once but it must always be of the same type. Plot statements can display numeric, true-false, and string expressions, but they cannot change types within an analysis technique. For example, the following pair of *Plot* statements are not allowed in an analysis technique because they include different data types, where the first plot is a text string and the second a true-false value:

```
Plot1 ("This is a text string");
If Condition1 Then
   Plot1 (Condition1);
```

302 "Different number of dimensions specified in the array than the parameter."

This error is shown when an array is passed into a function with the wrong number of dimensions. For example, this error will be generated if a function is expecting a single-dimension array but is sent an array with two dimensions instead.

303 "Extraneous text is not allowed after the array-type parameter."

When passing an array into a function, only the array name should be used. This error is displayed whenever any text, words, or braces are added after the array name that is passed to a function. For example:

```
Array: MyArray[10](0);
Value1 = Average_a(MyArray[0], 10);
```

the *[* will be highlighted because an array index appears after the array name. The correct syntax would be:

```
Array: MyArray[10](0);
Value1 = Average_a(MyArray, 10);
```

304 "Numeric-Array Parameter expected here."

Functions can receive arrays as parameters. If a function is expecting an array, any other type of parameter (variable, input, or reserved word) will display this error. In the following example:

```
Variable: MyVar(0);
Value1 = Average_a(MyVar, 10);
```

the function *Average_a()* requires an array on which to calculate an average and displays this error because *MyVar* is not an array.

Instead, you can write:

```
Array: MyArray[20](0);
Value1 = Average_a(MyArray, 10);
```

305 "TrueFalse-Array Parameter expected here."

Functions can now receive arrays as parameters. If a function is expecting a true-false array and, instead, the PowerEditor finds a variable, input, or other reserved word (different from a true-false array), it will display this error. For example, a function *MyTrueFalse_a()* that correctly uses true-false arrays would be written as follows:

```
Array: MyArray[20](False);
Variable: MyTF(False);

MyTF = MyTruefalse_a(MyArray, 10);
```

306 "String Array Parameter expected here."

Functions can now receive arrays as parameters. If a function is expecting an array of text strings and, instead, the PowerEditor finds a variable, input, or other reserved word (different from an array of text strings), it will display this error. For example, a function *Average_a()*, which combines all the text strings that are in an array into one, should be used as follows:

```
Array: MyArray[20](" ");
Variable: MyText(" ");
MyText = Average_a(MyArray, 10);
```

307 "The word 'Cancel' must be followed by 'Alert'."

Whenever canceling a previously enabled alert, the statement *Cancel Alert* needs to be used. This error is displayed whenever using the word *Cancel* without the word *Alert*.

314 "This word is only allowed in ActivityBar studies."

The words that are used to set the properties and draw ActivityBars are allowed only from ActivityBars and are not allowed in any other study or strategy.

323 "'Value-type inputs' may not be passed into 'reference-type inputs'."

Functions can receive array and variable parameters by reference or by value. However, if a function receives a variable or array by value, it is not possible to pass the parameter to a second function by reference. If an input of a function needs to be passed by reference to another function, it must also be declared as a reference input.

325 "Only an array is allowed here."

Functions can receive arrays as parameters. If a function is expecting an array, any other type of parameter (variable, input, or reserved word) will display this error. In the following example:

```
Variable: MyVar(0);
Value1 = Average_a(MyVar, 10);
```

the function *Average_a()* requires an array on which to calculate an average and displays this error because *MyVar* is not an array.

Instead, you can write:

```
Array: MyArray[20](0);
Value1 = Average_a(MyArray, 10);
```

340 “This word is only allowed when defining array-type inputs.”

This error is displayed when creating a function input using any input-type (such as *NumericArray*, *NumericArrayref*) without fully qualifying the input with braces. For example, this creates an error:

```
Input: MyInput(StringArrayRef );
```

because it does not include the array length parameter in brackets after the array name. The correct syntax would be:

```
Input: MyInput[n](StringArrayRef);
```

341 “An array input word (NUMERICARRAY, STRINGARRAY, TRUEFALSEARRAY, NUMERICARRAYREF, STRINGARRAYREF, TRUEFALSEARRAYREF) was expected here.”

When declaring inputs that are meant to receive an array, one of the above words is expected as the input type. For example, this error will be displayed when declaring an input for a function using the following statement:

```
Input: MyArray[M,N](Numeric);
```

since the reserved word *Numeric* is not valid for declaring arrays. However, the following will verify successfully:

```
Input: MyArray[M,N](NumericArray);
```

342 “This word can only be used in a PaintBar study.”

This error occurs when you use the reserved word *PlotPaintBar* when writing anything other than a PaintBar study.

396 “This statement cannot specify an odd number of plots.”

This error is displayed when using the *PlotPaintBar* statement and specifying an odd number of plots. There are two possible uses for this statement, either specifying only a high and low value, or specifying high, low, open, and close markers. The correct syntax for the *PlotPaintBar* statement follows:

```
PlotPaintBar(High, Low, "PB");
```

or

```
PlotPaintBar(High, Low, Open, Close, "PB");
```

403 “Cannot implicitly convert String to Numerical.”

Whenever the PowerEditor expects a numerical expression and, instead, finds a text string expression, it will highlight the text string expression and display this message.

For example, the following statement will produce this error:

```
Variable: MyNumber("55");
Value1 = Close + MyNumber;
```

Instead, the following expression accomplishes the expected result because it first uses the keyword *StrToNum()* to convert a text string expression to a numeric value:

```
Variable: MyNumber("55");
Value1 = Close + StrToNum(MyNumber);
```

404 “Cannot implicitly convert String to TrueFalse.”

Whenever the PowerEditor expects a true-false expression and, instead, finds a text string expression, it will highlight the text string expression and will display this message.

For example, the following statement will produce the error:

```
Input: Text1("Yes"), Text2("No");
Condition1 = Text1;
```

because the input “Text1” was declared as a text value and cannot be assigned the true-false variable *Condition1*. Instead, the following statement is correct:

```
Input: Text1("Yes"), Text2("No"); Condition1 = (Text1 = Text2);
```

Notice that while both *Text1* and *Text2* are string values, the result of the comparison is a true-false value that is properly assigned to a true-false variable.

405 "Cannot implicitly convert TrueFalse to String."

Whenever the PowerEditor expects a text string expression and, instead, finds a true-false expression, it will highlight the true-false expression and display this message. In this example, *Condition1* is a true-false variable and cannot be directly combined with a string:

```
FileAppend("Output.txt", "This is a text string" + Condition1);
```

Instead, the following expression corrects the problem by creating a string value based on whether *Condition1* is true or false:

```
Variable: txt(" ");

If Condition1 Then
   txt = "true"
Else
   txt = "false";
FileAppend("Output.txt", "This is a text string" + txt);
```

406 "Cannot implicitly convert Numerical to String."

Whenever the PowerEditor expects a text string expression and, instead, finds a numerical expression, it will highlight the numerical expression and will display this message.

For example:

```
FileAppend("Output.txt", "This is text" + Value1);
```

displays an error when a numeric expression is found. Instead, the following expression will accomplish the expected results because it uses the keyword *NumToString()* to convert a numerical expression to a string:

```
FileAppend("Output.txt", "This is text" + NumToStr(Value1, 2));
```

407 "Cannot implicitly convert TrueFalse to Numerical."

Whenever the PowerEditor expects a numerical value and, instead, finds a true-false expression, it will highlight the expression and will display this message.

For example, the following statement will produce this error because the *Condition1* value is a true-false variable and cannot be assigned to the numeric variable *Value1*:

```
Value1 = Condition1;
```

408 "Cannot implicitly convert Numerical to TrueFalse."

Whenever the PowerEditor expects a true/false expression and, instead, finds a numerical expression, it will highlight the numerical expression and will display this message. For example, the following statement produces an error because the reserved word *Open* is a numeric value and not a true-false expression:

```
Condition1 = Open;
```

Instead, assign the numeric value *Open* to the numeric variable *Value1*:

```
Value1 = Open ;
```

Or, change the statement such that it is a comparison. For example:

```
Condition1 = Open > Close;
```

Notice that while both *Open* and *Close* are numerical values, the result of the comparison is a true-false value, which is properly assigned to a true-false variable.

409 "String expression expected here."

This error is displayed whenever the PowerEditor is expecting a string expression and, instead, it finds a numeric or true-false expression. For example, this error will be displayed when writing information to a file with a *FileAppend* statement:

```
FileAppend("file.txt", Value1);
```

that includes the numeric expression *Value1* instead of a text string. Numeric expressions can be converted to strings by using the *NumToStr()* keyword. For example:

```
FileAppend("file.txt", NumToStr(Value1,2));
```

569 "Buy or SellShort name within double quotes expected here."

When specifying the name of a trading strategy, only a text string literal can be used, and it can't be substituted by a variable or an input. The following statements will generate this error:

```
Variable: txt("MyStrategy");
Buy (txt) Next Bar at Market;
```

whereas the correct way of assigning a name to a strategy is to use a literal string, such as:

```
Buy ("Strategy Name") Next Bar at Market;
```