SELString

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers
# Table of Contents

## Section 1: SELString
- Introduction ................................................................. 3
- Supported Firmware Versions ........................................... 4
- Global Parameters .......................................................... 4
- Functions ......................................................................... 4
- Classes .............................................................................. 5
- Benchmarks ....................................................................... 25
- Examples ........................................................................... 34
- Release Notes ................................................................. 45
SELString

Introduction

The purpose of this library is to provide a single object for performing string manipulations. It is intended to allow for a variety of string manipulations without requiring lengths to be predefined.

References to ASCII characters within this document are given in base-10. *Whitespace* for the scope of this document includes the following: spaces (32), tabs (9), and newlines (10). *Invalid Characters* are any characters that fall in the ASCII range 0–8, 11–27, or ≥127.

This document refers to iterator methods in a state called *locked out*. This refers to the state of the object being such that the user cannot retrieve non-NULL(0) values from *Next()* or *Previous()* without a new call to *Begin()* or *End()* or *Position()*.

Special Considerations

➢ To improve performance, this library manages a preallocated memory block for characters and SELString objects. To ensure proper functionality of the objects here, the *Clear()* methods of both SELString and SELStringList objects must be called before that object goes out of scope. For example, an SELString declared in the header of a function must have *Clear()* called on it before the end of the function. Variables instanti-ated in programs or global variable lists have permanent scope and do not need to be cleared unless it is desired to empty the data from them.

➢ This library expects that all SELString objects are used exclusively within a single IEC 61131 task. Using this library to instantiate objects on multiple tasks will create undesired behavior. For example, if you instantiate a class_SELString on the Main task of an RTAC, do not also instantiate a class_SELString on the automation task.

➢ Copying a class_SELString will cause undesired behavior. This means:

   1. The assignment operator ":=" must not be used on a class_SELString. Consider assigning a pointer to the class_SELString instead.

   2. class_SELString objects must never be VAR_INPUT or VAR_OUTPUT members in function blocks, functions, or methods. Place them in the VAR_IN_OUT section or use pointers instead.
This library can return NULL(0) pointers. Any pointer returned by the library should be validated to be non-NULL(0) before use, as shown in Code Snippet 5.

Supported Firmware Versions

You can use this library on any device configured using ACCELERATOR RTAC® SEL-5033 Software with firmware version R143 or higher.

Versions 3.5.0.3 and older can be used on RTAC firmware version R132 and higher.

Global Parameters

The library applies the following values as maximums; they can be modified when the library is included in a project.

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g_p_MaxIec61131StringSize</code></td>
<td>UINT</td>
<td>255</td>
<td>The largest size IEC 61131 string that this library can convert from/to.</td>
</tr>
<tr>
<td><code>g_p_SELStringInitialSize</code></td>
<td>UDINT</td>
<td>80</td>
<td>The number of bytes to initially allocate for a class_SELString.</td>
</tr>
</tbody>
</table>

Functions

fun_SELString_IsValidChar (Function)

This function takes a character in the form of a BYTE and returns FALSE if it is an invalid character; otherwise, this function returns TRUE.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>BYTE</td>
<td>Character to be evaluated for being valid.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE if char is a valid character.</td>
</tr>
</tbody>
</table>
Processing

If *char* is an invalid character, this function returns FALSE; otherwise, this function returns TRUE.

fun_SELString_IsWhitespace (Function)

This function takes a character in the form of a BYTE and returns TRUE if it is a whitespace character; otherwise, this function returns FALSE.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>BYTE</td>
<td>Character to be evaluated for being whitespace.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE if <em>char</em> is a whitespace character.</td>
</tr>
</tbody>
</table>

Processing

> If *char* is a whitespace character, this function returns TRUE; otherwise, this function returns FALSE.

Classes

class_SELString (Class)

class_SELString exists as the single class required to perform string manipulations. This class maintains a sequence of characters and is initialized to an empty set of characters.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>UDINT</td>
<td>R</td>
<td>The number of characters in this class_SELString.</td>
</tr>
</tbody>
</table>

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).
FromString (Method)

This method takes an IEC 61131 string, \( str \), as an input to be stored within the class_SELString.

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>STRING(g_p_MaxIec61131StringSize)</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

**Processing**

The FromString method performs the following:

- Accepts any valid IEC 61131 string up to \( g_p_{MaxIec61131StringSize} \) characters in size.
- Accepts all characters from the beginning of \( str \) until an invalid character, an end-of-string character, or \( g_p_{MaxIec61131StringSize} \) characters are reached; and no following characters.
- Replaces all current characters within class_SELString with characters from \( str \).
- Populates all available characters with the leading characters in \( str \) if the library runs out of memory. In this case, this method returns a pointer to an error string.
- Returns 0 on success.

FromByteArray (Method)

This method takes a pointer to the beginning of a byte array and the number of bytes to be copied. It then stores the byte array within the class_SELString object. This method stores any values represented in the byte array—it does not check for valid characters.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_byteArray</td>
<td>POINTER TO BYTE</td>
<td>The address returned by calling ADR() on the byte array to copy data from.</td>
</tr>
<tr>
<td>numBytes</td>
<td>UDINT</td>
<td>The number of bytes to be copied from the byte array.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

Processing

The `FromByteArray` method performs the following:

- Accepts all characters from the beginning of `pt_byteArray` until `numBytes` bytes.
- Replaces all current characters within this class_SELString with bytes from `pt_byteArray`.
- Populates all available characters with the leading bytes in `pt_byteArray` if the library runs out of memory and returns a pointer to an applicable error string.
- Returns 0 on success.

ToString (Method)

This method outputs an IEC 6131 string representation of the data being stored within the SELString.

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING(g_p_MaxIec61131StringLength)</td>
<td>An IEC 61131 string equivalent of the data stored within the SELString.</td>
</tr>
</tbody>
</table>

Processing

The `ToString` method:

- Returns an IEC 6131 string, up to `g_p_MaxIec61131StringLength` characters in size, that represents the first `g_p_MaxIec61131StringLength` characters of this class_SELString.
- If nothing has been assigned to this class_SELString, then the return value is an empty IEC 6131 string.
- The character immediately following the last character inserted into the IEC 6131 string is a string terminator character.

ToByteArray (Method)

This method copies the contents of the class_SELString to the address provided. It stops copying once `maxBytes` is reached or the class_SELString is completely copied.
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_destination</td>
<td>POINTER TO BYTE</td>
<td>The destination to which the content is copied.</td>
</tr>
<tr>
<td>maxBytes</td>
<td>UDINT</td>
<td>The maximum number of bytes to copy.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDINT</td>
<td>The number of bytes successfully copied to the destination.</td>
</tr>
</tbody>
</table>

**Processing**

The ToByteArray method:

- Writes the contents of the class_SELString to the address provided.
- Copying stops when one of the following occurs:
  1. The number of bytes copied is equal to maxBytes.
  2. All bytes in the class_SELString have been copied.
- The number of bytes copied before encountering one of the above criteria is returned as an integer. This will be 0 if class_SELString is empty or maxBytes is 0.
- No termination byte is appended.

**Replace (Method)**

This method replaces every occurrence of the string before with the string after.

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>before</td>
<td>class_SELString</td>
<td>The class_SELString containing the combination of characters to be replaced.</td>
</tr>
<tr>
<td>after</td>
<td>class_SELString</td>
<td>The class_SELString containing the combination of characters that will replace all characters of before.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

**Processing**

The Replace method:

- Does not modify before or after.
Is case sensitive.

Replaces all occurrences of before with after in a single pass. This prevents an infinite loop in cases like before="abc" and after="abcabc" and produces a reliable output in cases like before="abcabcdef" and after="bcb".

If before or after is this object, then this method leaves all objects unchanged. It then returns a pointer to an applicable IEC 61131 error string.

If the library runs out of memory, this method exits immediately in an undefined state. It returns a pointer to an applicable IEC 61131 error string.

Returns 0 on success.

Split (Method)

This method splits the class_SELString into multiple class_SELStrings wherever sep occurs and places the result in a class_SELStringList.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sep</td>
<td>class_SELString</td>
<td>The combination of characters defining where the string will split.</td>
</tr>
<tr>
<td>stringlist</td>
<td>class_SELStringList</td>
<td>The string list to be populated.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

Processing

The Split method:

- Creates divisions at every occurrence of sep. For example:
  - In “tgeadadagt”, using a sep of “ada” results in “tge”, “dagt”.
  - In “tgeadadagt”, using a sep of “ad” results in “tge”, “”, “agt”.
  - In “decide”, using a sep of “de” results in “”, “ci”, “”.
- Is case sensitive.
- If the class_SELString was never assigned a string or contains an empty string, it fills stringlist with only an empty string in the first position.
- Erases any prior contents of stringlist.
- If sep does not exist in this class_SELString, it places the full class_SELString into the first position.
- If sep is empty, it places each character of this class_SELString into its own string. Example: “Hello” becomes: “H”, “e”, “l”, “l”, “o”.
- Adds substrings to stringlist in order from left to right.
Empties this class_SELString object.

If `sep` is this object, leaves all objects unchanged. It then returns a pointer to an applicable IEC 61131 error string.

If `sep` or this object exist within `stringlist`, it leaves all objects unchanged. It then returns a pointer to an applicable IEC 61131 error string.

If the library runs out of memory, it assigns as many class_SELStrings to `stringlist` as available, clearing itself of all characters that were transferred. All characters not transferred to new class_SELStrings remain in this class_SELString. It then returns a pointer to an applicable IEC 61131 error string.

Returns 0 on success.

Trim (Method)

This method removes excess whitespace from this class_SELString. In this method, whitespace includes any character that results in TRUE being returned from `fun_SELString_IsWhitespace`.

Processing

The Trim method performs the following:

- Removes all whitespace from the beginning of the string to the first non-whitespace character.
- Removes all whitespace from the last non-whitespace character to the end of the string.
- Replaces any number of consecutive whitespace characters with a single space character.
- Leaves a class_SELStrings containing no whitespace and empty strings unchanged.
- Replaces strings containing only whitespace with an empty string.

Append (Method)

This method appends the input, `str`, onto the end of this class_SELString. The input, `str`, is empty upon completion of this method.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>class_SELString</td>
<td>The class_SELString containing the characters that will be appended to this class_SELString.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>
Processing

The Append method:

- Adds all characters from \textit{str} after the last character of this class\_SELString.
- Removes all characters from \textit{str}.
- If this class\_SELString is empty, moves the characters from \textit{str} to this class\_SELString.
- If \textit{str} is empty, changes nothing in either class\_SELString.
- Changes nothing if \textit{str} is this object. It then returns a pointer to an applicable IEC 61131 error string.
- Always appends as many characters as possible. If the library runs out of memory, \textit{str} is empty upon completion and it returns a pointer to an applicable IEC 61131 error string.
- Returns 0 on success.

AppendString (Method)

This method appends the characters of the input, \textit{str}, onto the end of this class\_SELString.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{str}</td>
<td>STRING(g_p_MaxIec61131StringSize)</td>
<td>The IEC 61131 string containing the characters that will be appended to this class_SELString.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

Processing

The AppendString method:

- Adds all characters from \textit{str} after the last character of this class\_SELString.
- If this class\_SELString is empty, adds the characters from \textit{str} to this class\_SELString.
- If \textit{str} is empty, changes nothing in this class\_SELString.
- Always appends as many characters as possible. If the library runs out of memory, it returns a pointer to an applicable IEC 61131 error string.
- Returns 0 on success.
Prepend (Method)

This method prepends the input, `str`, onto the beginning of this `class_SELString`. The input, `str`, is empty upon completion of this method.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str</code></td>
<td><code>class_SELString</code></td>
<td>The <code>class_SELString</code> containing the characters to be prepended to this <code>class_SELString</code>.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

Processing

The Prepend method:

- Adds all characters from `str` before the first character of this `class_SELString`.
- Removes all characters from `str`.
- If this `class_SELString` is empty, moves the characters from `str` to this `class_SELString`.
- If `str` is empty, changes nothing in either `class_SELString`.
- Changes nothing if `str` is this object. It then returns a pointer to an applicable IEC 61131 error string.
- If the library runs out of memory, leaves all objects unchanged and returns a pointer to an applicable IEC 61131 error string.
- Returns 0 on success.

PrependString (Method)

This method prepends the characters of the input, `str`, to the beginning of this `class_SELString`.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str</code></td>
<td>STRING(g_p_MaxIec61131StringLength)</td>
<td>The IEC 61131 string containing the characters to be prepended to this <code>class_SELString</code>.</td>
</tr>
</tbody>
</table>
Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

Processing

The PrependString method:

- Adds all characters from \( str \) before the first character of this class_SELString.
- If this class_SELString is empty, adds the characters from \( str \) to this class_SELString.
- If \( str \) is empty, changes nothing in this class_SELString.
- If the library runs out of memory, leaves all objects unchanged and returns a pointer to an applicable IEC 61311 error string.
- Returns 0 on success.

Insert (Method)

This method inserts the input class_SELString, \( str \), into this class_SELString at the index specified by the user. The input \( str \) is empty upon completion of this method.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>UDINT</td>
<td>The index where ( str ) will be inserted.</td>
</tr>
</tbody>
</table>

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( str )</td>
<td>class_SELString</td>
<td>The class_SELString containing the characters that will be inserted to this class_SELString.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

Processing

The Insert method:

- Inserts at index, where an index of zero corresponds to immediately before the first character and is incremented by one for each character thereafter.
- Removes all characters from \( str \).
- Inserts all characters from \( str \) into this class_SELString at position index.
- If `index` is larger than the size of this class_SELString, appends `str` at the end of this class_SELString.
- If this class_SELString is empty, moves the characters from `str` to this class_SELString.
- If `str` is empty, changes nothing in either class_SELString.
- Changes nothing if `str` is this object. It then returns a pointer to an applicable IEC 61131 error string.
- If the library runs out of memory, places all characters of `str` into this class_SELString, truncates the result, and returns a pointer to an applicable IEC 61131 error string.
- Returns 0 on success.

## Find (Method)

This method returns the first position after the provided `index` where the character combination provided in `str` exists. If the character combination does not exist after the provided `index`, a –1 is returned.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>UDINT</td>
<td>The index where this class_SELString will begin to search.</td>
</tr>
</tbody>
</table>

### Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>class_SELString</td>
<td>The class_SELString containing the characters that are being searched for within this class_SELString.</td>
</tr>
</tbody>
</table>

### Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT</td>
<td>The index of the first match of <code>str</code> within this object. If <code>str</code> cannot be found, this method will return ‘–1’. (This value is always ≥ ‘–1’)</td>
</tr>
</tbody>
</table>

### Processing

The Find method:

- Begins its search at `index`, where an `index` of zero corresponds to immediately before the first character and is incremented by one for each character thereafter. The same is true of the return value.
- Is case sensitive.
- Returns the index of the first occurrence of `str` appearing after the given index.
- If `index` is larger than the size of this class_SELString, returns –1.
If \( \text{str} \) cannot be found, returns \(-1\).
If \( \text{str} \) is empty, returns \(-1\).
If \( \text{str} \) is this object, then \(-1\) is returned.

**FindString (Method)**

This method returns the first position after the provided \( \text{index} \) where the character combination provided in \( \text{str} \) exists. If the character combination does not exist after the provided \( \text{index} \), a \(-1\) is returned.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{index} )</td>
<td>UDINT</td>
<td>The index where this class_SELString will begin to search.</td>
</tr>
</tbody>
</table>

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{str} )</td>
<td>STRING(g_p_MaxIec61131StringSize)</td>
<td>The IEC 61131 string containing the characters that are being searched for within this class_SELString.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT</td>
<td>The index of the first match of ( \text{str} ) within this object. If ( \text{str} ) cannot be found, this method will return (-1). (This value is always ( \geq -1 ))</td>
</tr>
</tbody>
</table>

**Processing**

The FindString method:

- Begins its search at \( \text{index} \), where an \( \text{index} \) of zero corresponds to immediately before the first character and is incremented by one for each character thereafter. The same is true of the return value.
- Is case sensitive.
- Returns the index of the first occurrence of \( \text{str} \) appearing after the given index.
- If \( \text{index} \) is larger than the size of this class_SELString, returns \(-1\).
- If \( \text{str} \) cannot be found, returns \(-1\).
- If \( \text{str} \) is empty, returns \(-1\).
- If \( \text{str} \) system state prevents the search, then \(-1\) is returned.
Clear (Method)

This method removes all character data from the class_SELString.

Processing

The Clear method:
- Removes all character data within the class_SELString.
- Resets the size of the class_SELString to zero.
- Resets the return value of ToString to an empty string.
- Any used memory is returned to the library.

Item (Method)

This method returns the character at the requested index.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>UDINT</td>
<td>The index of the character being requested.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>The ASCII value of the character at the requested index. If index is out of range, returns NULL(0).</td>
</tr>
</tbody>
</table>

Processing

The Item method:
- Treats the first character as index zero with each additional character being an addition of one.
- Returns the character positioned at the user requested index.
- If index is greater than or equal to the number of characters in this class_SELString, returns NULL.
- Returns NULL on an empty string.

Begin (Method)

This method is used in conjunction with Next() and Previous(). This method places the internal iterator on the first character of this class_SELString object.
Processing

The Begin method places the iterator such that:

- The iterator is not locked out.
- A subsequent `Next()` returns the first character.
- A subsequent `Previous()` returns NULL and leaves the iterator locked out.
- For an empty string, `Next()` and `Previous()` return NULL and leave the iterator locked out.

End (Method)

This method is used in conjunction with `Next()` and `Previous()`. This method places the internal iterator immediately after the last character of this class_SELString object.

Processing

The End method places the iterator such that:

- The iterator is not locked out.
- A subsequent `Previous()` returns the last character.
- A subsequent `Next()` returns NULL and leaves the iterator locked out.
- For an empty string `Next()` and `Previous()` return NULL and leave the iterator locked out.

Position (Method)

This method is used in conjunction with `Next()` and `Previous()`. This method places the internal iterator on the character at index.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>UDINT</td>
<td>The location where the character iterator will be assigned.</td>
</tr>
</tbody>
</table>

Processing

The Position method places the iterator such that:

- The first character in the class_SELString is `index` zero with each additional character being an addition of one.
- An `index` greater than or equal to the number of characters in this class_SELString leaves the iterator locked out.
- Otherwise the iterator is not locked out.
- A subsequent `Next()` returns the character at the provided `index`.
- A subsequent `Previous()` returns the character immediately before the provided `index`. 
For an empty string `Next()` and `Previous()` return NULL and leave the iterator locked out.

**Next (Method)**

This method is used in conjunction with `Begin()`, `Position()`, `End()`, and `Previous()`. This method returns the character at the current internal iterator position then increments the iterator.

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>The ASCII value of the character at the current iterator position. If no character exists, returns NULL(0).</td>
</tr>
</tbody>
</table>

**Processing**

The `Next` method:

- Returns the character at the current internal iterator position and then increments the iterator.
- Returns NULL if the iterator is locked out.
- Locks out the iterator for subsequent calls to `Next()` and `Previous()` if the iterator increments to a position greater than the number of characters in this class SELString.
- Locks out the iterator if any method other than `Next()` or `Previous()` has been called since the last `Begin()`, `End()`, or `Position()`.

**Previous (Method)**

This method is used in conjunction with `Begin()`, `Position()`, `End()`, and `Next()`. This method decrements the current internal iterator position and then returns the character at the new position.

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>The ASCII value of the character at the decremented iterator position. If no character exists, returns NULL(0).</td>
</tr>
</tbody>
</table>

**Processing**

The `Next` method performs the following:

- Decrement the internal iterator position then returns the character new position.
- Returns NULL if the iterator is locked out.
- Locks out the iterator for subsequent calls to `Next()` and `Previous()` if the iterator decrements to a position less than index 0.
Locks out the iterator if any method other than Next() or Previous() has been called since the last Begin(), End(), or Position().

**class_SELStringList (Class)**

class_SELStringList exists to store and perform operations on a list of strings using 0-based indexing.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>UDINT</td>
<td>R</td>
<td>The number of class_SELString objects in this class_SELStringList.</td>
</tr>
</tbody>
</table>

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

**Append (Method)**

This method adds the requested string to the end of the list.

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>class_SELString</td>
<td>The string to be added to the end of the class_SELStringList.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

**Processing**

The Append method:

- Places str at the end of the current list (position zero for an empty list) and increments the size of the list.
- Empties the contents of str.
- Allows empty strings to be added to the list.
- If the object str is already a member of the list, empties the contents of str and places the contents of str at the end of the list.

For example: If this list is [“dog”,”cat”,”bird”] and Item 1 (“cat”) is appended, the new list is: [“dog”,””,”bird”,”cat”].
If the library runs out of memory, it leaves this object and $str$ unchanged and returns a pointer to an applicable IEC 61131 error string.

- Returns 0 on success.

**Insert (Method)**

This method inserts the requested string into the list at the position specified.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>UDINT</td>
<td>The location where $str$ will be inserted.</td>
</tr>
</tbody>
</table>

**Inputs/Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$str$</td>
<td>class_SELString</td>
<td>The class_SELString to be inserted into the class_SELStringList.</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO STRING</td>
<td>A pointer to an error string or zero on success.</td>
</tr>
</tbody>
</table>

**Processing**

The Insert method:

- Places the $index$ zero immediately before the first class_SELString and increments by one for each string thereafter.
- Empties the contents of $str$.
- Allows empty strings to be added to the list.
- When $index$ is equal to the length of list or more, behaves the same as Append().
- If the object $str$ is already a member of the list, empties the contents of $str$ and places the contents of $str$ at the requested location.
  
  For example: If this list is [“dog”, “cat”, “bird”] and Item 0 (“dog”) is inserted at Position 2, the new list is: [“”, “cat”, “dog”, “bird”].
- Increases the size of the list by one.
- If the library runs out of memory, removes the final class_SELString from this object and inserts $str$ at the desired $index$ except in the case where $str$ would be appended, in which case the Insert method behaves the same as Append(). It then returns a pointer to an applicable IEC 61131 error string.
- Returns 0 on success.
RemoveLast (Method)

This method removes the last string in the list.

Processing

The RemoveLast method performs the following:
- Removes the last class_SELString from the class_SELStringList.
- Decrements the size of the class_SELStringList.
- Does nothing if the class_SELStringList is empty.
- Returns the freed memory.

Clear (Method)

This method removes all class_SELString objects from the list.

Processing

The Clear method performs the following:
- Removes all class_SELStrings in the class_SELStringList.
- Sets the size of the class_SELStringList to zero.
- Causes all other methods to respond that there are no objects in the list.
- Returns the freed memory.

Item (Method)

This method returns a pointer to the class_SELString at the requested index.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>UDINT</td>
<td>The location of the desired string in the list.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO class_SELString</td>
<td>A pointer to the class_SELString located at index. If index does not exist, returns NULL(0).</td>
</tr>
</tbody>
</table>

Processing

The Item method performs the following:
- Returns a pointer to the class_SELString that exists at index.
Returns NULL if index is outside the bounds of the list.

**Begin (Method)**

This method is used in conjunction with Next() and Previous(). This method places the internal iterator on the first string in the list.

**Processing**

The Begin method places the iterator such that:

- The iterator is not locked out.
- A subsequent Next() returns the first string in the list.
- A subsequent Previous() returns NULL and leaves the iterator locked out.
- For an empty list Next() and Previous() return NULL and leave the iterator locked out.

**End (Method)**

This method is used in conjunction with Next() and Previous(). This method places the internal iterator immediately after the last string in the list.

**Processing**

The End method places the iterator such that:

- The iterator is not locked out.
- A subsequent Next() returns NULL and leaves the iterator locked out.
- A subsequent Previous() returns the first string in the list.
- For an empty list Next() and Previous() return NULL and leave the iterator locked out.

**Position (Method)**

This method is used in conjunction with Next() and Previous(). This method places the internal iterator on the string at index.

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>UDINT</td>
<td>List location of the desired string.</td>
</tr>
</tbody>
</table>
Processing

The Position method places the iterator such that:

- The first string in the class_SELStringList is index zero with each additional string being an addition of one.
- An index greater than or equal to the number of strings in this class_SELStringList leaves the iterator locked out.
- Otherwise the iterator is not locked out.
- A subsequent Next() returns the string at the provided index.
- A subsequent Previous() returns the string immediately before the provided index.
- For an empty string list, Next() and Previous() return NULL and leave the iterator locked out.

Next (Method)

This method is used in conjunction with Begin(), Position(), End(), and Previous().
This method returns the string at the current internal iterator position then increments the iterator.

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO class_SELString</td>
<td>A pointer to the string at the current iterator position. If no string exists, returns NULL(0).</td>
</tr>
</tbody>
</table>

Processing

The Next method performs the following:

- Returns the string at the current internal iterator position and then increments the iterator.
- Returns NULL if the iterator is locked out.
- Locks out the iterator for subsequent calls to Next() and Previous() if the iterator increments to a position greater than the number of strings in this class_SELStringList.
- Locks out the iterator if any method other than Next() or Previous() has been called since the last Begin(), End(), or Position().

Previous (Method)

This method is used in conjunction with Begin(), Position(), End(), and Next().
This method decrements the internal iterator position and returns a pointer to the string at the new position.
Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINTER TO class_SELString</td>
<td>A pointer to the string at the decremented iterator position. If no string exists, returns NULL(0).</td>
</tr>
</tbody>
</table>

Processing

The Previous method performs the following:

- Decrements the internal iterator position and returns a pointer to the string at the new position.
- Returns NULL if the iterator is locked out.
- Locks out the iterator for subsequent calls to Next() and Previous() if the iterator decrements below position 0.
- Locks out the iterator if any method other than Next() or Previous() has been called since the last Begin(), End(), or Position().

Join (Method)

This method creates a single string containing the full list of strings, each separated from the other by the provided str. This object is empty upon successful completion.

Inputs/Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>class_SELString</td>
<td>The class_SELString to be inserted between each entry in the class_SELStringList.</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class_SELString</td>
<td>Returns a string containing all strings within the stringlist separated by str.</td>
</tr>
</tbody>
</table>

Processing

The Join method:

- Returns a single string containing the full list of strings separated by str.
  Example: If the class_SELStringList contains: ["b","n","n","s"] and str is “a” then the return value is “bananas”.
- Returns an empty string, if empty.
- Returns a single string containing the full list of strings if str is an empty string.
  For example: If the stringlist contains [“H”,“e”,“l”,“l”,“o”] and str is empty, the return value is “Hello”.
- Empties this class_SELStringList as per Clear().
If `str` is a member of this list, returns a valid `class_SELString` with undefined values.

If the library runs out of memory, returns a `class_SELString` containing all possible complete `class_SELStrings` in the list (strings will not be divided). This object retains any strings not used in the Join.

---

**Benchmarks**

**Benchmark Platforms**

The benchmarking tests recorded for this library are performed on the following platforms.

- **SEL-3530-4**
  - R132 firmware

- **SEL-3354**
  - Intel Pentium 1.4 GHz
  - 1 GB DDR ECC SDRAM
  - SEL-3532 RTAC Conversion Kit
  - R132 firmware

**Benchmark Test Descriptions class_SELString**

The posted time for each test is the average execution time of 50 consecutive calls for the test as described. The maximum number of characters used in any test was `g_p_StringSize`. All times are recorded in microseconds.

**FromString Replace**

This test calls `FromString()` and replaces `g_p_StringSize` characters with `g_p_StringSize` different characters.

**FromString Populate**

This test calls `FromString()` on an empty `class_SELString` and populates it with `g_p_StringSize` characters.

**FromString Depopulate**

This test calls `FromString()` on a `class_SELString` with `g_p_StringSize` characters and populates it with zero characters.
** SELString Instruction Manual Date Code 20180925 **

### Benchmarks

**ToString**

This test calls `ToString()` on a class `SELString` with `g_p_StringSize` characters.

**FromByteArray Replace**

This test calls `FromString()` and replaces `g_p_StringSize` characters with `g_p_StringSize` different characters.

**FromByteArray Populate**

This test calls `FromString()` on an empty `class_SELString` and populates it with `g_p_StringSize` characters.

**FromByteArray Depopulate**

This test calls `FromString()` on a `class_SELString` with `g_p_StringSize` characters and populates it with zero characters.

**ToByteArray**

This test calls `ToString()` on a `class_SELString` with `g_p_StringSize` characters.

**Replace 1 Char Not Found**

This test calls `Replace()` with a `before` of length one in a `class_SELString` of length `g_p_StringSize` without finding the result. It is designed to show the cost of a one-pass search of a string.

**Replace 1 Char Max Found**

This test calls `Replace()` with a `before` of length one in a `class_SELString` of length `g_p_StringSize`, finding the result on every character. It is designed to show the cost of a one-pass search and replace on a string.

**Replace Max/2 Found**

This test calls `Replace()` with a `before` of “aaaaaaab” and current of “aaaaaaaaaaab” but with sizes of `g_p_StringSize/2` and `g_p_StringSize` respectively. It is designed to show the cost of heavy recursive searching on a string.
Replace Max Char Found

This test calls \texttt{Replace()} with a \textit{before} of “aaaaaaab” and current of “aaaaaaab” but with sizes of \texttt{g\_p\_StringSize}. It is designed to show the cost of a one-pass ordered search on a string.

Split 0 Char

This test calls \texttt{Split()} with an empty separator in a \texttt{class\_SELString} of length \texttt{g\_p\_StringSize}. It is designed to show the cost of splitting a string into its constituent characters.

Split 1 Char Not Found

This test calls \texttt{Split()} with a one character separator in a length \texttt{g\_p\_StringSize} \texttt{class\_SELString} without finding the result. It is designed to show the cost of a one-pass search conversion of a string to a string list.

Split 1 Char Found

This test calls \texttt{Split()} with a one character separator in a length \texttt{g\_p\_StringSize} \texttt{class\_SELString}, finding the result in every character. It is designed to show the cost of converting a complete list to empty strings.

Split Max/2 Found

This test calls \texttt{Split()} with a \texttt{sep} of “aaaaaaab” and current of “aaaaaaaaaaaaab” but with sizes of \texttt{g\_p\_StringSize}/2 and \texttt{g\_p\_StringSize} respectively. It is designed to show the cost of heavy recursive searching and character removal using \texttt{Split()}.

Split Max Found

This test calls \texttt{Split()} with a \texttt{sep} of “aaaaaaab” and current of “aaaaaaab” but with sizes of \texttt{g\_p\_StringSize}. It is designed to show the cost of a one-pass ordered search and character removal on a string using \texttt{Split()}.

Trim No Whitespace

This test calls \texttt{Trim()} on a length \texttt{g\_p\_StringSize} \texttt{class\_SELString} containing no whitespace. It is designed to show the cost of searching for whitespace.

Trim Every Other Whitespace

This test calls \texttt{Trim()} on a length \texttt{g\_p\_StringSize} \texttt{class\_SELString} where every other character is whitespace. It is designed to show the cost of searching for and ignoring whitespace.
Trimm All Whitespace

This test calls Trim() on a length g_p_StringSize class_SELString containing only whitespace. It is designed to show the cost of removal of whitespace.

Trimm Half Whitespace

This test calls Trim() on a length g_p_StringSize class_SELString containing whitespace to remove at the head, in the middle, and at the end of the class_SELString. It is designed to show a more composite cost of whitespace removal.

Size

This test calls Size() on a length g_p_StringSize class_SELString.

Append

This test calls Append() with varying length class_SELString.

AppendString

This test calls AppendString() with a g_p_MaxIec61131StringSize length string.

Prepend

This test calls Prepend() with varying length class_SELString.

PrependString

This test calls PrependString() with a g_p_MaxIec61131StringSize length string.

Insert Either End

This test calls Insert() on the ends of a class_SELString.

Insert Middle

This test calls Insert() to place one character in the middle of a class_SELString.

Find 1 Char

This test calls Find() for a single character not found in a length g_p_StringSize class_SELString.
Find Max/2

This test calls `Find()` searching for “aaaaaaab” while having a current of “aaaaaaaaaaaaab” but with sizes of `g_p_StringSize/2` and `g_p_StringSize` respectively. It is designed to show the cost of heavy recursive searching using `Find()`.

Find Max

This test calls `Find()` searching for a length `g_p_StringSize` string inside a length `g_p_StringSize` class `SELString`.

FindString 1 Char

This test calls `FindString()` for a single character not found in a length `g_p_StringSize` class `SELString`.

FindString Max/2

This test calls `FindString()` searching for “aaaaaaab” of size `g_p_StringSize/2` while having a current of “aaaaaaaaaaaaab” or size of `g_p_StringSize`. It is designed to show the cost of heavy recursive searching using `FindIecString()`.

FindString Max

This test calls `FindString()` searching for a length `g_p_StringSize` string inside a length `g_p_StringSize` class `SELString`.

Clear

This test calls `Clear()` on a `class_SELString` of length `g_p_StringSize`.

Item Either End

This test calls `Item()` on both ends of a length `g_p_StringSize` class `SELString`.

Item Middle

This test calls `Item()` requesting the middle of a length `g_p_StringSize` class `SELString`.

Begin

This test calls `Begin()`.
End

This test calls End() on a length g_p_StringSize class_SELString.

Position Either End

This test calls Position() on both ends of a length g_p_StringSize class_SELString.

Position Middle

This test calls Position() requesting the middle of a length g_p_StringSize class_SELString.

Next

This test calls Next() while both locked out and responsive.

Previous

This test calls Previous() while both locked out and responsive.

Benchmark Results class_SELString

<table>
<thead>
<tr>
<th>Operation Tested</th>
<th>Platform (time in µs)</th>
<th>SEL-3354</th>
<th>SEL-3530</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromString Replace</td>
<td></td>
<td>9</td>
<td>89</td>
</tr>
<tr>
<td>FromString Populate</td>
<td></td>
<td>45</td>
<td>274</td>
</tr>
<tr>
<td>FromString Depopulate</td>
<td></td>
<td>16</td>
<td>79</td>
</tr>
<tr>
<td>ToString</td>
<td></td>
<td>8</td>
<td>75</td>
</tr>
<tr>
<td>FromByteArray Replace</td>
<td></td>
<td>7</td>
<td>75</td>
</tr>
<tr>
<td>FromByteArray Populate</td>
<td></td>
<td>42</td>
<td>241</td>
</tr>
<tr>
<td>FromByteArray Depopulate</td>
<td></td>
<td>16</td>
<td>77</td>
</tr>
<tr>
<td>ToByteArray</td>
<td></td>
<td>7</td>
<td>65</td>
</tr>
<tr>
<td>Replace 1 Char Not Found</td>
<td></td>
<td>16</td>
<td>165</td>
</tr>
<tr>
<td>Replace 1 Char Max Found</td>
<td></td>
<td>70</td>
<td>701</td>
</tr>
<tr>
<td>Replace Max/2 Found</td>
<td></td>
<td>535</td>
<td>4701</td>
</tr>
<tr>
<td>Replace Max Char Found</td>
<td></td>
<td>19</td>
<td>181</td>
</tr>
<tr>
<td>Split 0 Char</td>
<td></td>
<td>103</td>
<td>633</td>
</tr>
<tr>
<td>Split 1 Char Not Found</td>
<td></td>
<td>17</td>
<td>176</td>
</tr>
<tr>
<td>Split 1 Char Found</td>
<td></td>
<td>127</td>
<td>951</td>
</tr>
<tr>
<td>Split Max/2 Found</td>
<td></td>
<td>534</td>
<td>4673</td>
</tr>
<tr>
<td>Split Max Found</td>
<td></td>
<td>25</td>
<td>169</td>
</tr>
<tr>
<td>Trim No Whitespace</td>
<td></td>
<td>11</td>
<td>87</td>
</tr>
<tr>
<td>Trim Every Other Whitespace</td>
<td></td>
<td>11</td>
<td>92</td>
</tr>
<tr>
<td>Trim All Whitespace</td>
<td></td>
<td>29</td>
<td>206</td>
</tr>
<tr>
<td>Operation Tested</td>
<td>Platform (time in µs)</td>
<td>SEL-3354</td>
<td>SEL-3530</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Trim Half Whitespace</td>
<td></td>
<td>20</td>
<td>155</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Append</td>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>AppendString</td>
<td></td>
<td>49</td>
<td>272</td>
</tr>
<tr>
<td>Prepend</td>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>PrependString</td>
<td></td>
<td>45</td>
<td>276</td>
</tr>
<tr>
<td>Insert Either End</td>
<td></td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Insert Middle</td>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Find 1 Char</td>
<td></td>
<td>16</td>
<td>170</td>
</tr>
<tr>
<td>Find Max/2</td>
<td></td>
<td>521</td>
<td>4617</td>
</tr>
<tr>
<td>Find Max</td>
<td></td>
<td>8</td>
<td>71</td>
</tr>
<tr>
<td>FindString 1 Char</td>
<td></td>
<td>16</td>
<td>164</td>
</tr>
<tr>
<td>FindString Max/2</td>
<td></td>
<td>567</td>
<td>4796</td>
</tr>
<tr>
<td>FindString Max</td>
<td></td>
<td>67</td>
<td>430</td>
</tr>
<tr>
<td>Clear</td>
<td></td>
<td>16</td>
<td>77</td>
</tr>
<tr>
<td>Item Either End</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Item Middle</td>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Begin</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>End</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Position Either End</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Position Middle</td>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Next</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Previous</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Benchmark Test Descriptions class_SELStringList**

The posted time for each test is the average execution time of 50 consecutive calls for the test as described. The maximum number of characters used in any test was g_p_StringSize. A maximum size class_SELStringList tested was also g_p_StringSize long, each string containing one character. All times were recorded in microseconds.

**Append**

This test calls Append() on a class_SELStringList of g_p_StringSize class_SELStrings.

**Insert Either End**

This test calls Insert() on both ends of a class_SELStringList of g_p_StringSize class_SELStrings.

**Insert Middle**

This test calls Insert() requesting the middle of a class_SELStringList of length g_p_StringSize.
**RemoveLast**

This test calls `RemoveLast()` on a class_SELStringList of `g_p_StringSize` class_SELStrings.

**Size**

This test calls `Size()` on a class_SELStringList of `g_p_StringSize` class_SELStrings.

**Clear 1 Large String**

This test calls `Clear()` on a class_SELStringList of one class_SELString of length `g_p_StringSize`.

**Clear Max Strings**

This test calls `Clear()` on a class_SELStringList containing the `g_p_StringSize` number of one character class_SELStrings.

**Item Either End**

This test calls `Item()` requesting both ends of a class_SELStringList containing the `g_p_StringSize` number of one character class_SELStrings.

**Item Middle**

This test calls `Item()` requesting the middle of a class_SELStringList containing `g_p_StringSize` number of one character class_SELStrings.

**Begin**

This test calls `Begin()`.

**End**

This test calls `End()` on a class_SELStringList of `g_p_StringSize` class_SELStrings.

**Position Either End**

This test calls `Position()` requesting each end of a class_SELStringList containing `g_p_StringSize` class_SELStrings.
Position Middle

This test calls `Position()` requesting the middle class SELString of a class SELStringList containing `g_p_StringSize` class SELStrings.

Next

This test calls `Next()` while both locked out and responsive.

Previous

This test calls `Previous()` while both locked out and responsive.

Join Max Empties

This test calls `Join()` with a `str` of length one on a class SELStringList containing `g_p_StringSize` empty strings.

Join Max 1 Chars

This test calls `Join()` with a `str` of length zero on a class SELStringList containing `g_p_StringSize` one character strings.

This test calls `Join()` with a `str` of length `g_p_StringSize` on a class SELStringList containing two empty strings.

Benchmark Results class SELStringList

<table>
<thead>
<tr>
<th>Operation Tested</th>
<th>Platform (time in µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SEL-3354</td>
</tr>
<tr>
<td>Append</td>
<td>1</td>
</tr>
<tr>
<td>Insert Either End</td>
<td>1</td>
</tr>
<tr>
<td>Insert Middle</td>
<td>2</td>
</tr>
<tr>
<td>RemoveLast</td>
<td>17</td>
</tr>
<tr>
<td>Size</td>
<td>1</td>
</tr>
<tr>
<td>Clear 1 Large String</td>
<td>15</td>
</tr>
<tr>
<td>Clear Max Strings</td>
<td>59</td>
</tr>
<tr>
<td>Item Either End</td>
<td>1</td>
</tr>
<tr>
<td>Item Middle</td>
<td>1</td>
</tr>
<tr>
<td>Begin</td>
<td>1</td>
</tr>
<tr>
<td>End</td>
<td>1</td>
</tr>
<tr>
<td>Position Either End</td>
<td>1</td>
</tr>
<tr>
<td>Position Middle</td>
<td>1</td>
</tr>
<tr>
<td>Next</td>
<td>1</td>
</tr>
<tr>
<td>Previous</td>
<td>1</td>
</tr>
<tr>
<td>Join Max Empties</td>
<td>100</td>
</tr>
</tbody>
</table>
### Examples

These examples demonstrate the capabilities of this library. Do not mistake them as suggestions or recommendations from SEL.

Implement the best practices of your organization when using these libraries. As the user of this library, you are responsible for ensuring correct implementation and verifying that the project using these libraries performs as expected.

### class_SELString Examples

The following examples demonstrate simple uses of class_SELString.

#### FromString

**Code Snippet 1  SELString FromString Example**

```plaintext
PROGRAM prg_FromStringExample
VAR
  normalString : STRING := 'This is a normal string';
  selString : class_SELString;
END_VAR

// Populate the selString with the contents of normalString.
selString.FromString(normalString);
```

#### ToString

**Code Snippet 2  SELString ToString Example**

```plaintext
PROGRAM prg_ToStringExample
VAR
  normalString : STRING := 'This is a string';
  selString : class_SELString;
END_VAR

// Populate the SELString.
selString.FromString('This is a string');
// Converts the SELString to a normal string.
normalString := selString.ToString();
```
FromByteArray ToByteArray

If data is being read-in from, or out to a communication buffer, it is likely to contain non-printable characters that are deliberately ignored by the FromString and ToString methods. This example shows how to accomplish the same thing, using the FromByteArray and ToByteArray methods, which are able to accept all byte values.

Code Snippet 3  SELString To/From ByteArray Example

```plaintext
PROGRAM prg_ToAndFromByteArrayExample
VAR CONSTANT
c_StringLength : UDINT := 80;
END_VAR
VAR
InputString : STRING(c_StringLength) := 'This is a normal string';
OutputString : STRING(c_StringLength);
NumberBytesRead : UDINT;
SelString : class_SELString;
END_VAR

// Populate the SELString.
SelString.FromByteArray(ADR(InputString), c_StringLength);
// Converts the SELString to a byte array.
NumberBytesRead := SelString.ToByteArray(ADR(OutputString), c_StringLength);
// Append the null character on the end, since it is omitted by
// ToByteArray.
OutputString[c_StringLength] := 0;
```

Replace

Code Snippet 4  SELString Replace Example

```plaintext
PROGRAM prg_ReplaceExample
VAR
normalString : STRING := 'This is a normal string';
selString : class_SELString;
before : class_SELString;
after : class_SELString;
END_VAR

// Create an SELString with the text to be replaced.
before.FromString('normal');
// Create an SELString with the desired replacement text.
after.FromString('SEL');
// Populate the SELString.
selString.FromString(normalString);
// Replace the text in the SELString. This SELString now contains
// 'This is a SEL string'.
selString.Replace(before, after);
```
# Split

**Code Snippet 5  SELString Split Example**

```plaintext
PROGRAM prg_SplitExample
VAR
   normalString : STRING := 'This is a normal string';
   selString : class_SELString;
   selStringSpace : class_SELString;
   selStringList : class_SELStringList;

   pt_FirstString : POINTER TO class_SELString;
   pt_SecondString : POINTER TO class_SELString;

   firstString : STRING;
   secondString : STRING;
END_VAR

// Populate the SELString.
selString.FromString(normalString);
// Create an SELString containing the delimiter.
selStringSpace.FromString(' ');
// Split the SELString into a list of SELStrings, where each SELString in
// the list is a word from the original SELString.
selStringList.Split(selStringSpace, selStringList);
// Get the items from the list
pt_FirstString := selStringList.Item(0);
pt_SecondString := selStringList.Item(1);
// Dereference the pointers and convert to IEC 61131 Strings
// Check for a valid pointer before dereference
IF 0 <> pt_FirstString THEN
   firstString := pt_FirstString^.ToString();
ELSE
   ; // Error Message
END_IF
IF 0 <> pt_SecondString THEN
   secondString := pt_SecondString^.ToString();
ELSE
   ; // Error Message
END_IF
```

---

**Examples**

**Code Snippet 5 SELString Split Example**
Trim

Code Snippet 6  SELString Trim Example

```plaintext
PROGRAM prg_TrimExample
VAR
  selString : class_SELString;
END_VAR

// Populate the SELString.
selString.FromString("This string has excess whitespace.");
// Trim the extra whitespace, resulting in,
// 'This string has excess whitespace.'.
selString.Trim();
```

Size

Code Snippet 7  SELString Size Example

```plaintext
PROGRAM prg_SizeExample
VAR
  selString : class_SELString;
  size : UDINT;
END_VAR

// Populate the SELString.
selString.FromString("This is a string.");
// Get the size of the string.
size := selString.Size;
```

Append

Code Snippet 8  SELString Append Example

```plaintext
PROGRAM prg_AppendExample
VAR
  firstString : class_SELString;
  secondString : class_SELString;
END_VAR

// Populate the first string.
firstString.FromString("This is a ");
// Populate the second string.
secondString.FromString("string.");
// Append the second string to the first string.
firstString.Append(secondString);
```
Prepend

**Code Snippet 9  SELString Prepend Example**

```plaintext
PROGRAM prg_PrependExample
VAR
    firstString : class_SELString;
    secondString : class_SELString;
END_VAR

// Populate the first string.
firstString.FromString('This is a ');
// Populate the second string.
secondString.FromString('string.');
// Prepend the first string to the second string.
secondString.Prepend(firstString);
```

Insert

**Code Snippet 10  SELString Insert Example**

```plaintext
PROGRAM prg_InsertExample
VAR
    firstString : class_SELString;
    secondString : class_SELString;
END_VAR

// Populate the first string.
firstString.FromString('This is a string.');
// Populate the second string.
secondString.FromString('SEL ');
// Insert the first string into the second string.
firstString.Insert(10, secondString);
```

Find

**Code Snippet 11  SELString Find Example**

```plaintext
PROGRAM prg_FindExample
VAR
    firstString : class_SELString;
    secondString : class_SELString;
    index : DINT;
END_VAR

// Populate the first string.
firstString.FromString('This is an SEL string.');
// Populate the second string.
secondString.FromString('SEL ');
// Find the index of the string 'SEL' in the first string.
index := firstString.Find(0, secondString);
```
Clear

Code Snippet 12  SELString Clear Example

```plaintext
PROGRAM prg_ClearExample
VAR
    selString : class_SELString;
END_VAR

// Populate the string.
selString.FromString('This is an SEL string.');
// Clear the contents of the string.
selString.Clear();
```

Item

Code Snippet 13  SELString Item Example

```plaintext
PROGRAM prg_ItemExample
VAR
    selString : class_SELString;
    character : BYTE;
END_VAR

// Populate the string.
selString.FromString('This is an SEL string.');
// Get the ASCII value of the character located at index 11.
character := selString.Item(11);
```

Begin, End, Position, Next, and Previous

Code Snippet 14  SELString Iterator Example

```plaintext
PROGRAM prg_IteratorExample
VAR
    selString : class_SELString;
    firstCharacter : BYTE;
    middleCharacter : BYTE;
    lastCharacter : BYTE;
END_VAR

// Populate the string.
selString.FromString('This is an SEL string.');
// Get the first character.
selString.Begin();
firstCharacter := selString.Next();
// Get a middle character.
selString.Position(11);
middleCharacter := selString.Next();
// Get the last character
selString.End();
lastCharacter := selString.Previous();
```
class_SELStringList Examples

The following examples demonstrate simple uses of class_SELStringList.

Size

**Code Snippet 15  SELStringList Size Example**

```pascal
PROGRAM prg_SizeExample
VAR
  selString : class_SELString;
  selStringList : class_SELStringList;
  listSize : UDINT;
END_VAR

// Populate the string.
selString.FromString("This is an SEL string.");
// Append the string to the string list.
selStringList.Append(selString);
// Get the size of the string list.
listSize := selStringList.Size;
```

Append

**Code Snippet 16  SELStringList Append Example**

```pascal
PROGRAM prg_AppendExample
VAR
  selString : class_SELString;
  selStringList : class_SELStringList;
END_VAR

// Populate the string.
selString.FromString("This is an SEL string.");
// Append the string to the string list.
selStringList.Append(selString);
```

Insert

**Code Snippet 17  SELStringList Insert Example**

```pascal
PROGRAM prg_InsertExample
VAR
  selString1 : class_SELString;
  selString2 : class_SELString;
  selString3 : class_SELString;
  selStringList : class_SELStringList;
END_VAR
```
Code Snippet 17  SELStringList Insert Example (Continued)

// Populate the strings.
selString1FromString('String 1');
selString2FromString('String 2');
selString3FromString('String 3');

// Append the strings to the list.
selStringList.Append(selString1);
selStringList.Append(selString2);
// Insert a string to the list at index 1.
selStringList.Insert(1, selString3);

RemoveLast

Code Snippet 18  SELStringList RemoveLast Example

PROGRAM prg_RemoveLastExample
VAR
    selString1 : class_SELString;
    selString2 : class_SELString;
    selStringList : class_SELStringList;
END_VAR

// Populate the strings.
selString1FromString('String 1');
selString2FromString('String 2');

// Append the strings to the list.
selStringList.Append(selString1);
selStringList.Append(selString2);
// Remove the last string from the list.
selStringList.RemoveLast();

Clear

Code Snippet 19  SELStringList Clear Example

PROGRAM prg_ClearExample
VAR
    selString1 : class_SELString;
    selString2 : class_SELString;
    selStringList : class_SELStringList;
END_VAR
Examples

Code Snippet 19  SELStringList Clear Example (Continued)

```c
// Populate the strings.
selString1.FromString("String 1");
selString2.FromString("String 2");

// Append the strings to the list.
selStringList.Append(selString1);
selStringList.Append(selString2);
// Clear all strings from the list
selStringList.Clear();
```

Item

Code Snippet 20  SELStringList Item Example

```pascal
PROGRAM prg_ItemExample
VAR
  selStringList : class_SELStringList;
  firstString : class_SELString;
  middleString : class_SELString;
  lastString : class_SELString;
  ptFirstString : POINTER TO class_SELString;
  ptMiddleString : POINTER TO class_SELString;
  ptLastString : POINTER TO class_SELString;
END_VAR

// Populate the strings.
firstString.FromString("The first string.");
middleString.FromString("The middle string.");
lastString.FromString("The last string.");

// Add the strings to the list.
selStringList.Append(firstString);
selStringList.Append(middleString);
selStringList.Append(lastString);

// Get pointers to each of the strings.
ptFirstString := selStringList.Item(0);
ptMiddleString := selStringList.Item(1);
ptLastString := selStringList.Item(2);
```
Begin, End, Position, Next, and Previous

**Code Snippet 21**  SELStringList Iterator Example

```pascal
PROGRAM prg_IteratorExample
VAR
   selStringList : class_SELStringList;
   firstString : class_SELString;
   middleString : class_SELString;
   lastString : class_SELString;

   ptFirstString : POINTER TO class_SELString;
   ptMiddleString : POINTER TO class_SELString;
   ptLastString : POINTER TO class_SELString;
END_VAR

// Populate the strings.
firstString.FromString('The first string.');
middleString.FromString('The middle string.');
lastString.FromString('The last string.');

// Add the strings to the list.
selStringList.Append(firstString);
selStringList.Append(middleString);
selStringList.Append(lastString);

// Get the first string.
selStringList.Begin();
ptFirstString := selStringList.Next();
// Get the middle string.
selStringList.Position(1);
ptMiddleString := selStringList.Next();
// Get the last string.
selStringList.End();
ptLastString := selStringList.Previous();
```

Join

**Code Snippet 22**  SELStringList Join Example

```pascal
PROGRAM prg_JoinExample
VAR
   selStringList : class_SELStringList;
   firstString : class_SELString;
   middleString : class_SELString;
   lastString : class_SELString;
   delimiter : class_SELString;
   joinedString : class_SELString;
END_VAR
```
Code Snippet 22   SELStringList Join Example (Continued)

```csharp
// Populate the strings.
firstString.FromString("The");
middleString.FromString("joined");
lastString.FromString("string.");
delimiter.FromString(" ");

// Add the strings to the list.
selStringList.Append(firstString);
selStringList.Append(middleString);
selStringList.Append(lastString);

// Get the joined string, 'The joined string.'
joinedString := selStringList.Join(delimiter);
```
## Release Notes

<table>
<thead>
<tr>
<th>Version</th>
<th>Summary of Revisions</th>
<th>Date Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.0.1</td>
<td>➤ Initial release.</td>
<td>20140701</td>
</tr>
<tr>
<td>3.5.0.3</td>
<td>➤ Added FromByteArray and ToByteArray methods.</td>
<td>20140811</td>
</tr>
<tr>
<td></td>
<td>➤ Added AppendString and PrependString methods.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ Added FindString method.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ Protected classes against assignment.</td>
<td></td>
</tr>
<tr>
<td>3.5.1.0</td>
<td>➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types “Cannot convert” messages.</td>
<td>20180921</td>
</tr>
<tr>
<td></td>
<td>➤ Must be used with R143 firmware or later.</td>
<td></td>
</tr>
</tbody>
</table>