DynamicVectors

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers

Table of Contents

RTAC LIBRARY

DynamicVectors

Introduction

This library provides a vector data type for storing objects of various types, including arbitrary user-created objects. In general, a vector is an array of elements that is dynamically sized. As such, a vector allows elements to be added or removed and can contain an arbitrary number of elements. A vector also allows random access to its elements.

In addition to the vector classes, this library provides two factory functions for creating vectors: `fun_NewBaseVector()` and `fun_NewTypeVector()`. Calling a factory function creates a new object and returns a pointer to that object. For example, every call to `fun_NewBaseVector()` returns a pointer to a newly created class_BaseVector object.

See the ACSELERATOR RTAC Library Extensions Instruction Manual (LibraryExtensionsIM) for an explanation of the concepts used by the object-oriented extensions to the IEC 61131-3 standard.

Special Considerations

➤ Copying classes from this library causes unwanted behavior. This means the following:

1. The assignment operator ":=" must not be used on any class from this library; consider assigning pointers to the objects instead.

```
// This is bad and in most cases will provide a compiler error
    such as:
// "C0328: Assignment not allowed for type class_VectorObject"
myVectorObject := otherVectorObject;
```

```
// This is fine
someVariable := myVectorObject.value;
// As is this
pt_myVectorObject := ADR(myVectorObject);
```

2. Classes from this library 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.

➤ Classes in this library have memory allocated inside them. As such, they should only be created in environments of permanent scope (e.g., Programs, Global Variable Lists, or VAR_STAT sections).

Supported Firmware Versions

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

Versions 3.5.1.0 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.

Name	IEC 61131 Type	Value	Description
g_p_DefaultVectorSize	UDINT	32	The default number of elements that a vector can hold.

Enumerations

Enumerations make code more readable by allowing a specific number to have a readable textual equivalent.

enum_DynamicVectorType

This enumeration is used for specifying the desired type of vector to the `fun_NewVector()` function.

Enumeration	Description
BYTE_VECTOR	Enumeration for class_ByteVector.
WORD_VECTOR	Enumeration for class_WordVector.
DWORD_VECTOR	Enumeration for class_DwordVector.
LWORD_VECTOR	Enumeration for class_LwordVector.
REAL_VECTOR	Enumeration for class_RealVector.
LREAL_VECTOR	Enumeration for class_LrealVector.
POINTER_VECTOR	Enumeration for class_PointerVector.

Functions

fun_NewBaseVector (Function)

This function creates a new class_BaseVector and returns a pointer to the newly created vector. The returned POINTER TO BYTE must be cast to the correct type before it is used. A vector created with this function must be destroyed with `fun_DeleteVector()` when it is no longer needed.

Inputs

Name	IEC 61131 Type	Description
elementSize	UDINT	The size of each element in the vector.

Return Value

IEC 61131 Type	Description
POINTER TO BYTE	Pointer to the newly created vector. This pointer is null if the vector could not be created.

Processing

- Creates a new vector with the specified elementSize and returns a pointer to the newly created vector.
- Returns a null pointer if the vector could not be created.

fun_NewTypeVector (Function)

This function creates a new vector of the type specified and returns a pointer to the newly created vector. The returned POINTER TO BYTE must be cast to the correct type before it is used. A vector created with this function must be destroyed with `fun_DeleteVector()` when it is no longer needed.

Inputs

Name	IEC 61131 Type	Description
vectorType	enum_DynamicVectorType	The desired type of vector.

Return Value

IEC 61131 Type	Description
POINTER TO BYTE	Pointer to the newly created vector. This pointer is null if the vector could not be created.

Processing

- Creates a new vector of the type specified and returns a pointer to the newly created vector.
- Returns a null pointer if the vector could not be created.

fun_DeleteVector (Function)

This function deletes a vector created with `fun_NewBaseVector()` or `fun_NewTypeVector()`. After deletion, any pointers to the deleted vector are no longer valid.

Inputs

Name	IEC 61131 Type	Description
vector	I_Vector	The vector to delete.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if vector is successfully deleted. False if an error occurs.

Interfaces

This library provides the following interface.

I_Vector

This interface is implemented by any class that provides a vector data type.

Properties

Name	IEC 61131 Type	Access	Description
pt_BaseVector	POINTER TO class_BaseVector	R	Pointer to the class_BaseVector used internally by this vector.
pt_Data	POINTER TO BYTE	R	Pointer to the raw memory array used internally by this vector.
ElementSize	UDINT	R	The number of bytes required for each element in the vector.
MaxSize	UDINT	R	The number of elements this vector can currently hold before a reallocation for additional memory is required.

Properties

Name	IEC 61131 Type	Access	Description
pt_Self	POINTER TO BYTE	R	The THIS pointer for the class.
Size	UDINT	R	The number of elements in the vector.

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 appends an array of elements to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
pt_array	POINTER TO BYTE	Pointer to the first element to append to the vector.
numElements	UDINT	The number of elements to copy from the array.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the elements are successfully appended to the vector. FALSE if an error occurs.

Processing

- If pt_array is null, the vector is not modified and this method returns FALSE.
- If pt_array is valid and numElements is zero, the vector is not modified and this method returns TRUE.
- If appending to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns false.

Clear (Method)

Deallocates all memory associated with the vector. Call this method only if the vector is instantiated with limited scope (i.e., if it is instantiated as a local variable of a function or method).

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the vector successfully deallocates its internal memory. FALSE if an error occurs.

Recycle (Method)

This method removes all elements from the vector without modifying the memory allocated to the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the vector successfully removes all elements. FALSE if an error occurs.

Processing

All elements are removed from the vector.

- After recycling, the Size property of the vector is zero.
- The MaxSize property is unchanged after a call to `Recycle()`.
- This method neither allocates nor frees any memory.

Resize (Method)

This method resizes the vector so it can contain the number of elements specified without requiring any additional memory allocations.

Inputs

Name	IEC 61131 Type	Description
newSize	UDINT	The desired number of elements for the resized vector to contain without requiring a memory reallocation.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the vector is successfully resized. FALSE if an error occurs.

Processing

This method resizes the vector so it can contain the number of elements specified without requiring any additional memory allocations.

- If the specified newSize is zero, then the vector is resized to g_p_DefaultVectorSize or the present number of elements, whichever is greater.
- If the specified newSize is greater than zero and less than the present number of elements, the vector is resized to the present number of elements.
- If the specified newSize is greater than the number of elements, the vector is resized to the specified newSize.
- If the specified newSize equals the present maximum size of the vector, the vector is not resized and the method returns TRUE.

Classes

class_BaseVector

This class implements a generic vector that internally handles dynamic allocation of memory. This vector can handle objects of arbitrary size so long as the number of bytes required for each element is the same. This vector stores its internal data in a contiguous block of memory.

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

- I_Vector

Initialization Inputs

Name	IEC 61131 Type	Description
elementSize	UDINT	The number of bytes required for each element this vector will hold. If zero, then default to one.
numElements	UDINT	The number of elements to account for initially. If zero, then use the default.

GetCopyOfElement (Method)

This method copies the element at the index specified from the vector to the destination pointer.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the element to copy from the vector.
pt_destination	POINTER TO BYTE	A pointer to the destination to which the element is copied.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the specified element is copied. FALSE if an error occurs.

Processing

If index or pt_destination are invalid, nothing is copied and false is returned.

PopTo (Method)

This method copies the last element in the vector to the provided pointer location and then deletes the last element.

Inputs

Name	IEC 61131 Type	Description
pt_destination	POINTER TO BYTE	A pointer to the destination to which the element is copied.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the last element is successfully copied and removed from the vector. FALSE if an error occurs.

Processing

- Copies the last element in the vector to the provided pointer location.
- Removes the last element in the vector.
- If the vector does not contain any elements (i.e., the size is zero), the method returns false without modifying the vector.
- If pt_destination is invalid or the element cannot be copied, then the vector is not modified and false is returned.

PushFrom (Method)

Copies the bytes from the provided pointer location to a new element at the end of the vector. Allocates additional memory if the vector does not have enough memory to contain the new element.

Inputs

Name	IEC 61131 Type	Description
pt_source	POINTER TO BYTE	A pointer to the source from which the element is copied.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the new element is added to the vector. FALSE if an error occurs.

Processing

- Copies the element at pt_source to a new element at the end of the vector.
- If the vector Size is MaxSize before the addition of the new element, the vector allocates additional memory. If the memory allocation fails, FALSE is returned and the vector is not modified.
- If pt_source is invalid, FALSE is returned and the vector is not modified.

SetElement (Method)

This method sets the element in the vector, specified by the index, to the contents of the source pointer.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the vector element to set.
pt_source	POINTER TO BYTE	A pointer to the source from which the element is copied.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the specified element is set. FALSE if an error occurs.

Processing

- Sets the element in the vector at index to the contents of pt_source.
- If index or pt_source is invalid, FALSE is returned and the vector is not modified.

class_ByteVector

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Vector

GetAt (Method)

Provides a copy of the element at the specified index.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the desired element in the vector.

Outputs

Name	IEC 61131 Type	Description
element	BYTE	The element at the specified index. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if index is valid and the element is copied. FALSE if index is invalid or an error occurs.

Pop (Method)

This method provides a copy of the last item in the vector and removes that element from the vector.

Outputs

Name	IEC 61131 Type	Description
element	BYTE	A copy of the former last element in the vector. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully copied and removed from the vector. FALSE if the size is zero or an error occurs.

Push (Method)

This method appends a copy of the provided element to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
element	BYTE	The element to copy to the end of the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully added to the vector. FALSE if an error occurs.

Processing

If pushing element to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns FALSE.

SetAt (Method)

This method provides write access to any element within the vector.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index at which to set the value of an element.
value	BYTE	The new element value.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the element is successfully modified. If index is invalid, the vector is not modified and FALSE is returned.

class_WordVector

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Vector

GetAt (Method)

Provides a copy of the element at the specified index.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the desired element in the vector.

Outputs

Name	IEC 61131 Type	Description
element	WORD	The element at the specified index. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if index is valid and the element is copied. FALSE if index is invalid or an error occurs.

Pop (Method)

This method provides a copy of the last item in the vector and removes that element from the vector.

Outputs

Name	IEC 61131 Type	Description
element	WORD	A copy of the former last element in the vector. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully copied and removed from the vector. FALSE if the size is zero or an error occurs.

Push (Method)

This method appends a copy of the provided element to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
element	WORD	The element to copy to the end of the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully added to the vector. FALSE if an error occurs.

Processing

If pushing element to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns FALSE.

SetAt (Method)

This method provides write access to any element within the vector.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index at which to set the value of an element.
value	WORD	The new element value.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the element is successfully modified. If index is invalid, the vector is not modified and FALSE is returned.

class_DwordVector

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Vector

GetAt (Method)

Provides a copy of the element at the specified index.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the desired element in the vector.

Outputs

Name	IEC 61131 Type	Description
element	DWORD	The element at the specified index. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if index is valid and the element is copied. FALSE if index is invalid or an error occurs.

Pop (Method)

This method provides a copy of the last item in the vector and removes that element from the vector.

Outputs

Name	IEC 61131 Type	Description
element	DWORD	A copy of the former last element in the vector. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully copied and removed from the vector. FALSE if the size is zero or an error occurs.

Push (Method)

This method appends a copy of the provided element to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
element	DWORD	The element to copy to the end of the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully added to the vector. FALSE if an error occurs.

Processing

If pushing element to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns FALSE.

SetAt (Method)

This method provides write access to any element within the vector.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index at which to set the value of an element.
value	DWORD	The new element value.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the element is successfully modified. If index is invalid, the vector is not modified and FALSE is returned.

class_LwordVector

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Vector

GetAt (Method)

Provides a copy of the element at the specified index.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the desired element in the vector.

Outputs

Name	IEC 61131 Type	Description
element	LWORD	The element at the specified index. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if index is valid and the element is copied. FALSE if index is invalid or an error occurs.

Pop (Method)

This method provides a copy of the last item in the vector and removes that element from the vector.

Outputs

Name	IEC 61131 Type	Description
element	LWORD	A copy of the former last element in the vector. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully copied and removed from the vector. FALSE if the size is zero or an error occurs.

Push (Method)

This method appends a copy of the provided element to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
element	LWORD	The element to copy to the end of the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully added to the vector. FALSE if an error occurs.

Processing

If pushing element to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns FALSE.

SetAt (Method)

This method provides write access to any element within the vector.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index at which to set the value of an element.
value	LWORD	The new element value.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the element is successfully modified. If index is invalid, the vector is not modified and FALSE is returned.

class_RealVector

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Vector

GetAt (Method)

Provides a copy of the element at the specified index.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the desired element in the vector.

Outputs

Name	IEC 61131 Type	Description
element	REAL	The element at the specified index. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if index is valid and the element is copied. FALSE if index is invalid or an error occurs.

Pop (Method)

This method provides a copy of the last item in the vector and removes that element from the vector.

Outputs

Name	IEC 61131 Type	Description
element	REAL	A copy of the former last element in the vector. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully copied and removed from the vector. FALSE if the size is zero or an error occurs.

Push (Method)

This method appends a copy of the provided element to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
element	REAL	The element to copy to the end of the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully added to the vector. FALSE if an error occurs.

Processing

If pushing element to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns FALSE.

SetAt (Method)

This method provides write access to any element within the vector.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index at which to set the value of an element.
value	REAL	The new element value.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the element is successfully modified. If index is invalid, the vector is not modified and FALSE is returned.

class_LrealVector

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Vector

GetAt (Method)

Provides a copy of the element at the specified index.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the desired element in the vector.

Outputs

Name	IEC 61131 Type	Description
element	LREAL	The element at the specified index. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if index is valid and the element is copied. FALSE if index is invalid or an error occurs.

Pop (Method)

This method provides a copy of the last item in the vector and removes that element from the vector.

Outputs

Name	IEC 61131 Type	Description
element	LREAL	A copy of the former last element in the vector. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully copied and removed from the vector. FALSE if the size is zero or an error occurs.

Push (Method)

This method appends a copy of the provided element to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
element	LREAL	The element to copy to the end of the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully added to the vector. FALSE if an error occurs.

Processing

If pushing element to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns FALSE.

SetAt (Method)

This method provides write access to any element within the vector.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index at which to set the value of an element.
value	LREAL	The new element value.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the element is successfully modified. If index is invalid, the vector is not modified and FALSE is returned.

class_PointerVector

Implemented Interfaces

An interface defines a required set of functionality as methods and properties. As an implementer of any interface all methods and properties declared in that interface must exist as members of this class. This allows multiple generally unrelated classes to be used interchangeably for a specific feature set.

➤ I_Vector

GetAt (Method)

Provides a copy of the element at the specified index.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index of the desired element in the vector.

Outputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	The element at the specified index. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if index is valid and the element is copied. FALSE if index is invalid or an error occurs.

Pop (Method)

This method provides a copy of the last item in the vector and removes that element from the vector.

Outputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	A copy of the former last element in the vector. If the return value is FALSE, this value is undefined.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully copied and removed from the vector. FALSE if the size is zero or an error occurs.

Push (Method)

This method appends a copy of the provided element to the end of the vector.

Inputs

Name	IEC 61131 Type	Description
element	POINTER TO BYTE	The element to copy to the end of the vector.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if element is successfully added to the vector. FALSE if an error occurs.

Processing

If pushing element to the vector requires more memory than is currently available in the vector, the library allocates additional memory. If the memory allocation fails, the vector is not modified and this method returns FALSE.

SetAt (Method)

This method provides write access to any element within the vector.

Inputs

Name	IEC 61131 Type	Description
index	UDINT	The index at which to set the value of an element.
value	POINTER TO BYTE	The new element value.

Return Value

IEC 61131 Type	Description
BOOL	TRUE if the element is successfully modified. If index is invalid, the vector is not modified and FALSE is returned.

Benchmarks

Benchmark Platforms

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

- SEL-3530
  - R134 firmware
- SEL-3354
  - Intel Pentium 1.4 GHz
  - 1 GB DDR ECC SDRAM
  - SEL-3532 RTAC Conversion Kit
  - R132 firmware
- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R134 firmware

Benchmark Test Descriptions

As benchmarks are designed to provide more information about speed of operation in known environments, only tests on the five pre-configured vectors are recorded here. Vectors on objects of varying sizes may have varying performance.

All benchmark tests shall be performed 100 times with the average result recorded here. In an attempt to allow other usage of system memory, only one iteration of each test is run per scan.

Append

The time to append 1000 objects to the vector.

Clear

The time to clear a vector of 1000 objects.

GetAt

The worst case time for retrieving an arbitrary index out of a vector of length 1000.

SetAt

The worst case time for setting an arbitrary index in a vector of length 1000.

Push32

The performance of Push when *Size* = *MaxSize* = 32.

Push1024

The performance of Push when *Size* = *MaxSize* = 1024.

Pop

The performance of a pop of the 1000th element of a vector.

Recycle

The time to remove all data from a vector of 1000 objects.

Resize Down

The performance of a manual resize from 2048 elements to 32 elements.

Resize Up

The performance of a manual resize from 32 elements to 2048 elements.

NewVector

The performance of requesting a new vector of the desired type.

Benchmark Results

Operation Tested	Platform (time in μs)		
	SEL-3530	SEL-3354	SEL-3555
Append			
Byte	67	9	6
Dword	58	6	3
Lreal	88	10	4
Lword	63	9	4
Pointer	44	5	3

Operation Tested	Platform (time in μs)		
	SEL-3530	SEL-3354	SEL-3555
Real	57	5	3
Word	43	5	3
Clear			
Byte	41	4	3
Dword	40	4	2
Lreal	40	4	2
Lword	43	4	2
Pointer	39	4	2
Real	39	4	2
Word	37	4	2
GetAt			
Byte	17	2	1
Dword	6	2	1
Lreal	7	2	1
Lword	9	1	1
Pointer	6	1	1
Real	8	1	1
Word	6	2	1
New			
Byte	73	23	10
Dword	43	6	4
Lreal	40	6	4
Lword	41	6	4
Pointer	37	6	4
Real	37	5	4
Word	42	6	4
Pop			
Byte	4	1	1
Dword	5	1	1
Lreal	4	1	1
Lword	4	1	1
Pointer	4	1	1
Real	5	1	1
Word	4	1	1
Push1024			
Byte	22	4	3
Dword	23	4	3
Lreal	36	5	3
Lword	33	5	3
Pointer	23	4	3
Real	22	4	3
Word	27	4	3
Push32			
Byte	32	5	3
Dword	33	5	3
Lreal	33	5	3
Lword	35	5	3
Pointer	31	5	3

Operation Tested	Platform (time in μs)		
	SEL-3530	SEL-3354	SEL-3555
Real	30	5	3
Word	31	4	3
Recycle			
Byte	2	1	1
Dword	2	1	1
Lreal	3	1	1
Lword	2	1	1
Pointer	2	1	1
Real	2	1	1
Word	2	1	1
ResizeDown			
Byte	23	4	3
Dword	23	4	3
Lreal	23	4	3
Lword	21	4	3
Pointer	20	4	3
Real	22	4	3
Word	23	4	3
ResizeUp			
Byte	35	5	3
Dword	35	4	3
Lreal	28	4	3
Lword	24	4	3
Pointer	24	4	3
Real	24	4	3
Word	30	5	3
SetAt			
Byte	5	2	1
Dword	6	2	1
Lreal	7	1	1
Lword	5	2	1
Pointer	6	2	1
Real	7	1	1
Word	8	1	1

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.

Creating and Using a class_ByteVector

Objective

This example shows how a basic class_ByteVector is instantiated and used. *Code Snippet 1* shows some simple manipulations of a byte vector. These include appending an array to the vector, popping bytes off the end of the vector, and checking the size of the vector.

Solution

Code Snippet 1 prg_ByteVector

```
PROGRAM prg_ByteVector
VAR
    // Flag to only run the program once.
    initialized : BOOL := FALSE;
    // A dynamically sized vector of bytes.
    byteVector : DynamicVectors.class_ByteVector();
    // A fixed size array of bytes to append to the vector.
    appendArray : ARRAY[1..5] OF BYTE := [1, 2, 3, 4, 5];
    // A fixed size array of bytes to pop off the vector.
    popArray : ARRAY[1..5] OF BYTE;
    // The size of the vector after appending.
    sizeAfterAppend : UDINT;
    // The size of the vector after popping.
    sizeAfterPop : UDINT;
    // Loop counter.
    i : UINT;
END_VAR
```

Code Snippet 1 prg_ByteVector (Continued)

```
// Only run the program once.
IF NOT initialized THEN
    initialized := TRUE;

    // Append the array to the empty vector.
    byteVector.Append(ADR(appendArray), 5);
    // The size after appending five bytes is five.
    sizeAfterAppend := byteVector.Size;

    // Pop all five elements off the vector and store in an array.
    FOR i := 1 TO 5 DO
        byteVector.Pop(element => popArray[i]);
    END_FOR

    // The size after popping the bytes off the vector is zero.
    sizeAfterPop := byteVector.Size;
END_IF
```

Creating a class_ByteVector with fun_NewByteVector()

Objective

This example shows how a basic class_ByteVector is created when using the factory function `fun_NewTypeVector()`. *Code Snippet 2* shows some simple manipulations of a byte vector created with a factory. These include appending an array to the vector, popping bytes off the end of the vector, and checking the size of the vector.

Solution

Code Snippet 2 prg_ByteVectorFactory

```
PROGRAM prg_ByteVectorFactory
VAR
    // Flag to only run once.
    initialized : BOOL := FALSE;
    // A pointer to a dynamically sized vector of bytes.
    pt_byteVector : POINTER TO DynamicVectors.class_ByteVector();
    // A fixed size array of bytes to append to the vector.
    appendArray : ARRAY[1..5] OF BYTE := [1, 2, 3, 4, 5];
    // A fixed size array of bytes to pop off the vector.
    popArray : ARRAY[1..5] OF BYTE;
    // The size of the vector after appending.
    sizeAfterAppend : UDINT;
    // The size of the vector after popping.
    sizeAfterPop : UDINT;
    // Loop counter.
    i : UINT;
END_VAR
```

Code Snippet 2 prg_ByteVectorFactory (Continued)

```
// Only run the program once.
IF NOT initialized THEN
    initialized := TRUE;

    (* Create the byte vector with the factory. fun_NewTypeVector creates a
     * byte vector and returns a pointer to the new vector. *)
    pt_byteVector := fun_NewTypeVector(BYTE_VECTOR);

    // Append the array to the empty vector.
    pt_byteVector^.Append(ADR(appendArray), 5);
    // The size after appending five bytes is five.
    sizeAfterAppend := pt_byteVector^.Size;

    // Pop all five elements off the vector and store in an array.
    FOR i := 1 TO 5 DO
        pt_byteVector^.Pop(element => popArray[i]);
    END_FOR

    // The size after popping the bytes off the vector is zero.
    sizeAfterPop := pt_byteVector^.Size;
END_IF
```

Creating and Using a class_BaseVector

Objective

This example shows how a class_BaseVector is instantiated and used. *Code Snippet 4* shows some simple manipulations of a base vector. These include appending an array to the vector, popping elements off the end of the vector, and checking the size of the vector.

Assumptions

This example assumes that there is a user-specified IEC 61131 data type defined as shown in *Code Snippet 3*.

Code Snippet 3 struct_UserObject

```
TYPE struct_UserObject :
STRUCT
    name : STRING;
    value : INT;
END_STRUCT
END_TYPE
```

Solution

Code Snippet 4 prg_BaseVector

```
PROGRAM prg_BaseVector
VAR
    // Flag to only run once.
    initialized : BOOL := FALSE;
    (* A dynamically sized vector of struct_UserObects. This instantiates a
     * vector of elements, where each element requires
     * SIZEOF(struct_UserObject) bytes of memory and to reserve memory for
     * 32 elements before a memory allocation is required. *)
    baseVector : DynamicVectors.class_BaseVector(SIZEOF(struct_UserObject),
        32);
    // A fixed size array to append to the vector.
    appendArray : ARRAY[1..5] OF struct_UserObject := [
        (name := 'number 1', value := 1),
        (name := 'number 2', value := 2),
        (name := 'number 3', value := 3),
        (name := 'number 4', value := 4),
        (name := 'number 5', value := 5)
    ];
    // A fixed size array to pop off the vector.
    popArray : ARRAY[1..5] OF struct_UserObject;
    // The size of the vector after appending.
    sizeAfterAppend : UDINT;
    // The size of the vector after popping.
    sizeAfterPop : UDINT;
    // Loop counter.
    i : UINT;
END_VAR
```

```
// Only run the program once.
IF NOT initialized THEN
    initialized := TRUE;

    // Append the array to the empty vector.
    baseVector.Append(ADR(appendArray), 5);
    // The size after appending five bytes is five.
    sizeAfterAppend := baseVector.Size;

    // Pop all five elements off the vector and store in an array.
    FOR i := 1 TO 5 DO
        baseVector.PopTo(ADR(popArray[i]));
    END_FOR

    // The size after popping the bytes off the vector is zero.
    sizeAfterPop := baseVector.Size;
END_IF
```

Resizing a Vector

Objective

This example demonstrates resizing a vector. Resizing a vector changes how much memory the vector reserves for the addition of new elements.

Addition of elements to a vector when it does not have space forces the vector to allocate additional memory. Memory allocation is a relatively slow operation, so it can be useful to have extra memory reserved by the vector to avoid unnecessary memory allocations. If it is known how many elements the vector will store, it can be resized once to avoid multiple slow memory allocations.

Resizing a vector is also useful for releasing memory from the vector. If a vector contained 100 elements previously, but currently only contains 10, the vector will still reserve memory for the previously removed 90 elements. If the vector no longer needs to store 90 additional elements, that memory can be returned to the system. A vector will not automatically release memory back to the system.

Code Snippet 5 shows the resizing of a vector to return memory to the system.

Solution

Code Snippet 5 prg_VectorResize

```
PROGRAM prg_VectorResize
VAR
    // Flag to only run once.
    initialized : BOOL := FALSE;
    // A dynamically sized vector of bytes.
    byteVector : DynamicVectors.class_ByteVector();
    // A fixed size array of bytes to append to the vector.
    appendArray : ARRAY[1..5] OF BYTE := [1, 2, 3, 4, 5];
    // A fixed size array of bytes to pop off the vector.
    popArray : ARRAY[1..5] OF BYTE;
    // The size of the vector after appending.
    sizeAfterAppend : UDINT;
    // The size of the vector after popping.
    sizeAfterPop : UDINT;
    // The maximum vector size before resizing.
    maxSizeBeforeResize : UDINT;
    // The maximum vector size after resizing.
    maxSizeAfterResize : UDINT;
    // Loop counter.
    i : UINT;
END_VAR
```

Code Snippet 5 prg_VectorResize (Continued)

```
// Only run the program once.
IF NOT initialized THEN
    initialized := TRUE;
    // Append the array to the empty vector.
    byteVector.Append(ADR(appendArray), 5);
    // The size after appending five bytes is five.
    sizeAfterAppend := byteVector.Size;
    // Pop all five elements off the vector and store in an array.
    FOR i := 1 TO 5 DO
        byteVector.Pop(element => popArray[i]);
    END_FOR
    // The size after popping the bytes off the vector is zero.
    sizeAfterPop := byteVector.Size;
    // The maximum size before resizing is 32 elements.
    maxSizeBeforeResize := byteVector.MaxSize;
    // Resize the vector so it can hold 10 bytes before a memory allocation
    // is required.
    byteVector.Resize(10);
    (* The maximum size after resizing is 10 elements. The memory required
     * for 22 elements is returned to the system, but if the vector needs
     * to store more than 10 elements in the future, it will need to
     * allocate additional memory. *)
    maxSizeAfterResize := byteVector.MaxSize;
END_IF
```

Release Notes

Version	Summary of Revisions	Date Code
3.5.2.0	➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types "Cannot convert" messages. ➤ Must be used with R143 firmware or later.	20180921
3.5.1.0	➤ Added typed vectors for REAL, LREAL, LWORD, and POINTER.	20150511
3.5.0.2	➤ Initial release.	20140811