CrossTaskData

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers

Table of Contents

RTAC LIBRARY

CrossTaskData

Introduction

The purpose of this library is to take a mutual exclusion (mutex) implementation that uses the SysSem semaphore library and wrap it into single function block calls to provide simple function blocks. Mutex data region creation depends upon the size of a user-defined structure.

In computer science, mutex refers to a basic concurrency control requirement that the critical sections of two concurrent processes cannot occur simultaneously. This requirement prevents race conditions.

In this particular case, the mutex allows one task to write the data and another to read the data. The mutex guarantees that the data remain intact. This is important because Real-Time Automation Controller (RTAC) automation and main tasks run with different priorities. To illustrate this, imagine a structure containing a value "stVal" and a time stamp "t" and that a low-speed task is writing periodically to this value and time stamp. Then imagine that a high-speed task uses this information for some computation. If there were no mutex in place, the lower priority main task may write only the "strVal" and be interrupted by the higher priority automation task. The logic running on the automation task would then be reading a newly updated "stVal" for which there is a "t" from the previous "stVal" instead of the time stamp that should be associated with the value. The mutex protects the integrity of the data, ensuring complete transfer of information.

The library contains a preset number of reserved mutexes defined by a global parameter. Each mutex has an identification number (ID) that is an integer value between 1 and the number the global parameter defines. Writer function blocks each claim one of the IDs, define the size of the mutex, and then set aside memory for the mutex as necessary.

Initialization of the writer and reader function blocks results in verification to ensure that only a single writer function block and a single reader function block exist per mutex ID.

After initialization, the only input necessary for the function block is the address of the data set that you want to read from and write to.

Special Considerations

The writer function block should never be instantiated in a method or a function. Because memory allocation occurs in order to create the mutex, writer function blocks must be instantiated in a program or global variable list. Declaring a writer function block in either a function or a method causes undesired behavior.

Each mutex ID may be referenced by no more than one writer function block.

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 3.5.0.3 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.

Name	IEC 61131 Type	Value	Description
g_p_N_CrossTaskIDs	UDINT	10	The number of mutexes available for writer function blocks.

Function Blocks

fb_CrossTaskWrite

If data exist on one task to be shared with another, use this function block on the task publishing the data. Only one writing function block may place data in a given mutex.

Initialization Inputs

Name	IEC 61131 Type	Description
id	UDINT	The ID of the mutex that the writer will own (range [1..g_p_N_CrossTaskIDs]).
sizeOfStruct	UDINT	The size of the structure, as returned by the SIZEOF() function.

Inputs

Name	IEC 61131 Type	Description
pt_Struct	DWORD	The address of the variable from which to read, as returned by the ADR() function.

Outputs

Name	IEC 61131 Type	Description
Error	STRING(80)	Any internal errors display here as a human-readable string. The string is empty if no errors are present.

Processing

- You must call the function block body to obtain a lock on the mutex with the ID specified.
- If the mutex is already locked (indicating another operation is in progress), the operation waits until the lock is released and it obtains the mutex.
- After obtaining the mutex, this function block copies the data referenced by pt_Struct into protected, internal memory and the lock is released.

fb_CrossTaskRead

If data exist on one task to be shared with another, use this function block on the task reading the data. More than one reading function blocks may pull data from the same mutex.

Initialization Inputs

Name	IEC 61131 Type	Description
id	UDINT	The ID of the mutex from which this function block reads (range [1..g_p_N_CrossTaskIDs]).

Inputs

Name	IEC 61131 Type	Description
pt_Struct	DWORD	The address of the variable to which the function block writes the data, as returned by the ADR() function.

Outputs

Name	IEC 61131 Type	Description
Error	STRING(80)	Any internal errors display here as a human-readable string. The string is empty if no errors are present.

Processing

- You must call the function block body to obtain a lock on the mutex with the ID specified.
- If the mutex is already locked (indicating another operation is in progress), the operation waits until the lock is released and it obtains the mutex.
- After obtaining the mutex, the function block copies the data from protected, internal memory into the location referenced by pt_Struct and the lock is released.

Benchmarks

Benchmark Platforms

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

- SEL-3505
  - R135-V2 firmware
- SEL-3530
  - R135-V2 firmware
- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R135-V2 firmware

Benchmark Test Descriptions

fb_CrossTaskWrite10

The posted time is the average execution time of 1000 calls in which a lock is obtained and data are written into the mutex. This constitutes the worst case for this call. The variable moved between tasks is an IEC 61131 dataset containing 10 UDINTs.

fb_CrossTaskRead10

The posted time is the average execution time of 1000 calls in which a lock is obtained and data are read from the mutex. This constitutes the worst case for this call. The variable moved between tasks is an IEC 61131 dataset containing 10 UDINTs.

fb_CrossTaskWrite10000

The posted time is the average execution time of 1000 calls in which a lock is obtained and data are written into the mutex. This constitutes the worst case for this call. The variable moved between tasks is an IEC 61131 dataset containing 10000 UDINTs.

fb_CrossTaskRead10000

The posted time is the average execution time of 1000 calls in which a lock is obtained and data are read from the mutex. This constitutes the worst case for this call. The variable moved between tasks is an IEC 61131 dataset containing 10000 UDINTs.

Benchmark Results

Operation Tested	Platform (time in μs)		
	SEL-3505	SEL-3530	SEL-3555
fb_CrossTaskWrite10	13	8	1
fb_CrossTaskRead10	17	9	1
fb_CrossTaskWrite10000	800	450	4
fb_CrossTaskRead10000	780	450	4

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.

Moving Data From a High-Priority Task to a Low-Priority Task

Objective

You have a structure called *dt_myAStruct* containing data from a high-speed, high-priority task. You want to copy this data to a slow-speed, low-priority task.

Assumptions

You have declared your IEC 61131 datatype, *dt_myAStruct*; here we use the declaration found in *Code Snippet 1*.

Code Snippet 1 dt_myAStruct

```
TYPE dt_myAStruct :
STRUCT
    a : UDINT;
    b : BOOL;
END_STRUCT
END_TYPE
```

Solution

Create a writing program on the high-speed task, as shown in *Code Snippet 2*.

Code Snippet 2 prg_FastTask

```
PROGRAM prg_FastTask
VAR
    _FastA : dt_myAStruct;
    _FastWriter : fb_CrossTaskWrite (ID := 1, sizeOfStruct :=
        SIZEOF(dt_myAStruct));
    _ErrorA : STRING(80);
END_VAR
```

```
// Populate the variable "_FastA" here.
_FastA.a := 20;
_FastA.b := TRUE;
// Send the data to the mutex.
_FastWriter(pt_Struct := ADR(_FastA), Error => _ErrorA);
```

Create a reading program on the slow-speed task, as shown in *Code Snippet 3*.

Code Snippet 3 prg_SlowTask

```
PROGRAM prg_SlowTask
VAR
    _SlowA : dt_myAStruct;
    _SlowReader : fb_CrossTaskRead (ID := 1);
    _ErrorA : STRING(80);
    _TheAvar : UDINT;
    _TheBvar : BOOL;
END_VAR
```

```
// First, read in all the data from the mutex to the local variable.
_SlowReader(pt_Struct := ADR(_SlowA), Error => _ErrorA);
// Now use _SlowA structure information as you wish.
_TheAvar := _SlowA.a;
_TheBvar := _SlowA.b;
```

Moving Data From a Low-Priority Task to a High-Priority Task

Objective

You have a structure called *dt_myBStruct* containing data from a low-speed, low-priority task. You want to copy these data to a high-speed, high-priority task.

Assumptions

You have declared your IEC 61131 datatype, *dt_myBStruct*; here we use the declaration found in *Code Snippet 4*.

Code Snippet 4 dt_myBStruct

```
TYPE dt_myBStruct :
STRUCT
    a : STRING(32);
    b : INT;
    c : REAL;
END_STRUCT
END_TYPE
```

Solution

Create a writing program on the slow-speed task, as seen in *Code Snippet 5*.

Code Snippet 5 prg_SlowTask

```
PROGRAM prg_SlowTask
VAR
    _SlowB : dt_myBStruct;
    _SlowWriter : fb_CrossTaskWrite(ID := 2, sizeOfStruct :=
        SIZEOF(dt_myBStruct));
    _ErrorB : String(80);
END_VAR
```

```
// Populate the variable "_SlowB" here.
_SlowB.a := 'helloFastTask';
_SlowB.b := -5;
_SlowB.c := 6.25;

// Send the data to the mutex.
_SlowWriter(pt_Struct := ADR(_SlowB), Error => _ErrorB);
```

Create a reading program on the high-speed task, as seen in *Code Snippet 6*.

Code Snippet 6 prg_FastTask

```
PROGRAM prg_FastTask
VAR
    _FastB : dt_myBStruct;
    _FastReader : fb_CrossTaskRead(ID := 2);
    _ErrorB : STRING(80);
    _TheAvar : STRING(32);
    _TheBvar : INT;
    _TheCvar : REAL;
END_VAR
```

```
// First, read in all the data from the mutex to the local variable.
_FastReader(pt_Struct := ADR(_FastB), Error => _ErrorB);

// Now use _SlowB structure information as you wish.
_TheAvar := _FastB.a;
_TheBvar := _FastB.b;
_TheCvar := _FastB.c;
```

Release Notes

Version	Summary of Revisions	Date Code
3.5.1.0	➤ 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
3.5.0.3	➤ Copy operations between tasks have been optimized to be more efficient for large cross task data sets.	20160708
3.5.0.2	➤ Corrected catching of SysMem internal errors.	20140828
3.5.0.1	➤ Initial release.	20140701