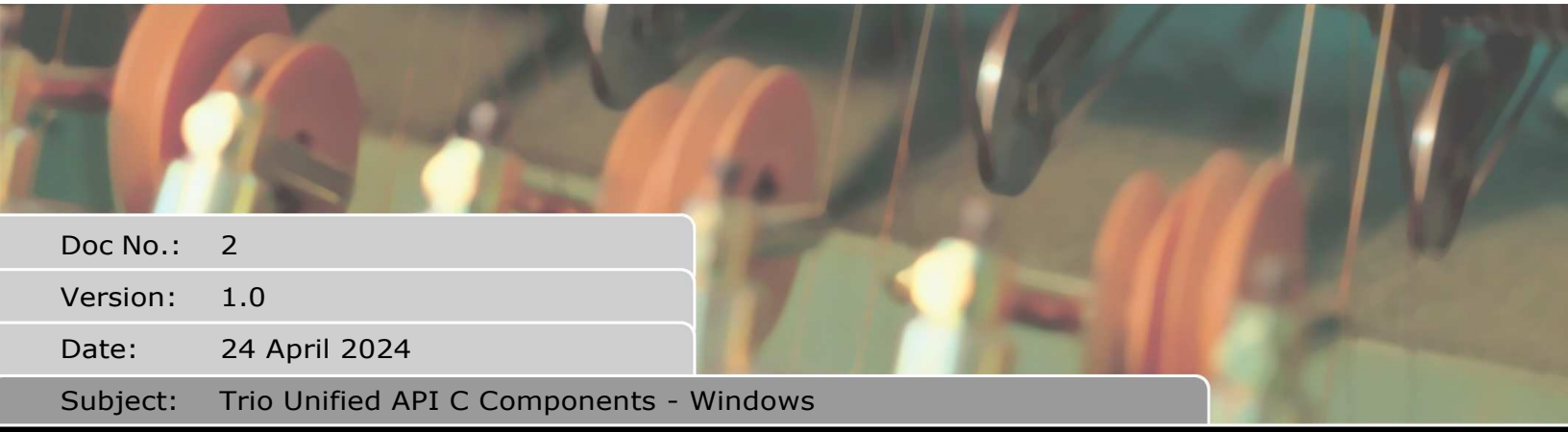
Trio Motion Technology Ltd.
Shannon Way, Tewkesbury,
Gloucestershire. GL20 8ND
United Kingdom
Tel: +44 (0)1684 292333
Fax: +44 (0)1684 297929

1000 Gamma Drive
Suite 206
Pittsburgh, PA 15238
United States of America
Tel: +1 412.968.9744
Fax: +1 412.968.9746

Tomson Centre
118 Zhang Yang Rd., B1701
Pudong New Area, Shanghai,
Postal code: 200122
P. R. CHINA
Tel/Fax: +86-21-58797659



Doc No.: 2
Version: 1.0
Date: 24 April 2024
Subject: Trio Unified API C Components - Windows

APPLICATION NOTE

Trio Unified API C Components - Windows

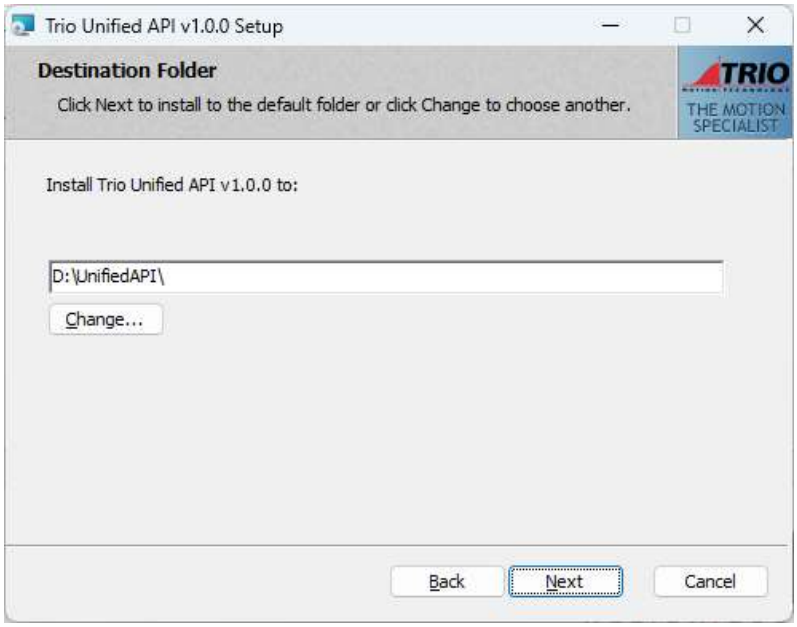
How to access controller via C using Unified API on Windows OS

1. Requirements

1.1. Windows 10 system (recommended)
1.2. Visual Studio 2022
1.3. Trio Unified API

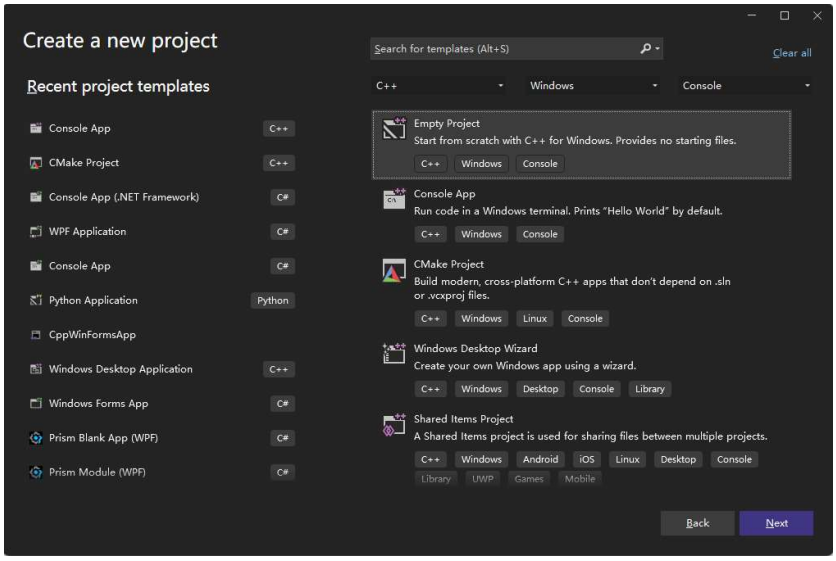
2. Project Setup

2.1. Install Trio Unified API in a folder on a local disk.

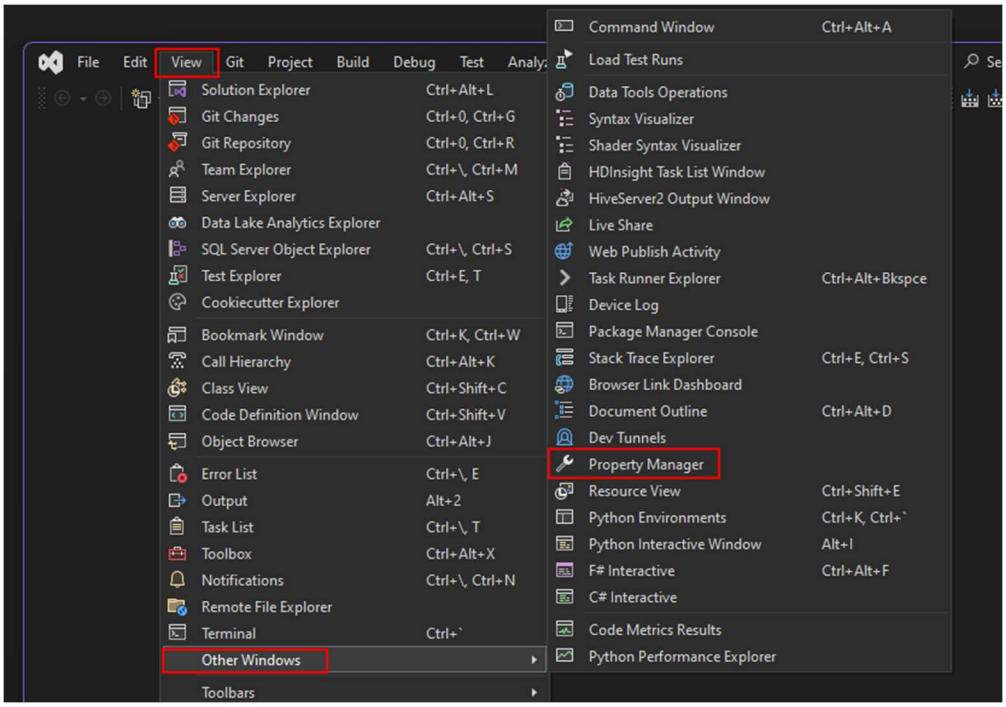


2.2. In this example will create a project in Visual Studio to test individual functions provided in Trio_UnifiedApi_C.h file.

2.3. Create an empty C project with the name <UAPI_Test_C> in a location on the windows disk.

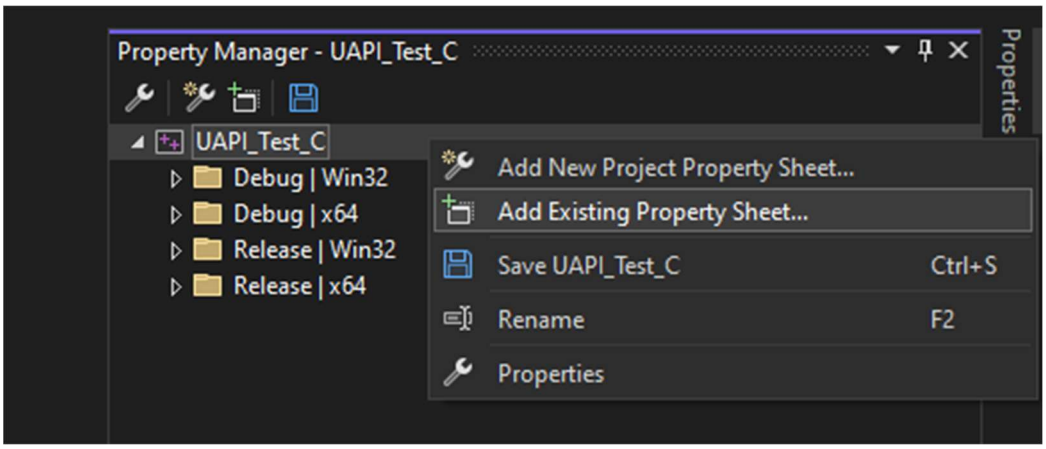


2.4. From Visual Studio menus navigate to 'View' -> 'Other Windows' and open 'Property Manager'.

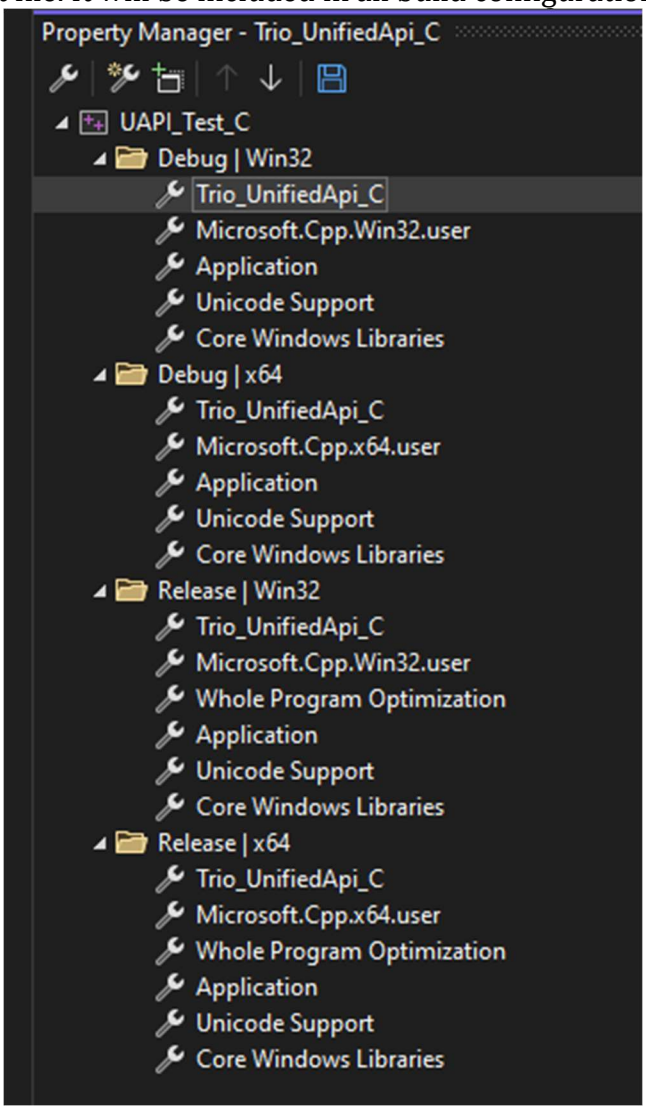


2.5. In 'Property Manager' right-click on 'UAPI_Test_C' and select 'Add Existing Property

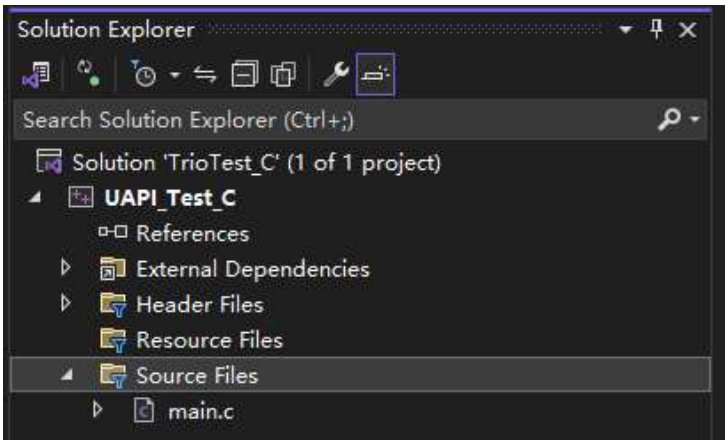
Sheet…’



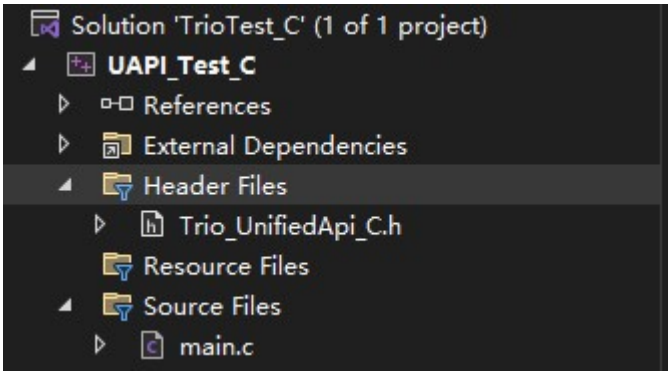
2.6. Navigate to Trio Unified API local folder and select ‘Trio_UnifiedApi_C.props’ property sheet file. It will be included in all build configurations.



2.7. Add a new C program file with the name <main.c> in the Source Files subfolder.



2.8. Add the Unified API header file to the project. Under Solution Explorer right click on Header Files and select “Add” -> “Existing Item…”. Browse and select Trio_UnifiedApi-C.h.



3. Example code

3.1. Include the Trio_UnifiedApi_C in the main.c.

```
#include <Trio_UnifiedApi_C.h>
```

3.2. Create the callback function.

```
// Connection callback
void TRIO_CALLCONV connection_callback(void* context, trio_EventType event_type, uint64_t
int_value, const char* str_value, size_t char_count)
{
        switch (event_type)
        {
        case trio_EventType_Error:
                printf("Error [%llx] occurred: %s\n", int_value, str_value);
                break;
```

```
        case trio_EventType_Message:
                printf("Msg: %s\n", str_value);
                break;

        default:
                break;
        }
}
```

3.3. Create the connection to the MC.

```
trio_ConnContext context = 0;
trio_ErrorCode res;
const char* mc_IP = "127.0.0.1"; // The IP address of the controller.
bool ok = true;

// Create connection context of TCP type
res = trio_CreateContextTCP2(connection_callback, 0, &context, mc_IP);
res = trio_OpenConnection(&context);
if (res == trio_ErrorCode_NoError)
{
        printf("Connected to the MC successfully\n");
}
```

3.4. Writing and reading variables, including the VR/TABLE/ AxisParameter/SystemParameter.

```
// Set VR value
double val = 125.25;
res = trio_SetVrValue(&context, 100, val);
if (res != trio_ErrorCode_NoError)
{
        printf("trio_SetVrValue returned error: %llx\n", res);
        ok = false;
}

//Get VR value
double valRd;
res = trio_GetVrValue(&context, 100, &valRd);
if (res != trio_ErrorCode_NoError)
{
        printf("trio_GetVrValue returned error: %llx\n", res);
        ok = false;
}
if (valRd != val)
{
        printf("trio_GetVrValue returned different value: %f != %f\n", val, valRd);
        ok = false;
}

//Set AxisParameter
int axis = 0;
double units = 100;
//AxisParam - UNITS
res = trio_SetAxisParameter_UNITS(&context, axis, units);
```

```
if (res != trio_ErrorCode_NoError)
{
	printf("trio_SetAxisParameter_UNITS returned error: %llx\n", res);
	ok = false;
}

//Set AxisParameter
float speed = 1000;
//AxisParam - SPEED
res = trio_SetAxisParameter_SPEED(&context, axis, speed);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_SetAxisParameter_SPEED returned error: %llx\n", res);
	ok = false;
}

//Get AxisParameter
double unitsRd;
//AxisParam - UNITS
res = trio_GetAxisParameter_UNITS(&context, axis, &unitsRd);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_GetAxisParameter_UNITS returned error: %llx\n", res);
	ok = false;
}
if (unitsRd != units)
{
	printf("trio_GetAxisParameter_UNITS returned different units: %f != %f\n",
units, unitsRd);
	ok = false;
}

//Get AxisParameter
double speedRd;
//AxisParam - SPEED
res = trio_GetAxisParameter_SPEED(&context, axis, &speedRd);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_GetAxisParameter_SPEED returned error: %llx\n", res);
	ok = false;
}
if (speedRd != speed)
{
	printf("trio_GetAxisParameter_SPEED returned different speed: %f != %f\n",
speed, speedRd);
	ok = false;
}

//Set system parameter
//SystemParam - WDOG
uint32_t wdog = 1;
res = trio_SetSystemParameter_WDOG(&context, wdog); //Turn the watchdog contact on(
enable the external drives)
if (res != trio_ErrorCode_NoError)
{
	printf("trio_SetSystemParameter_WDOG returned error: %llx\n", res);
	ok = false;
}
```

```
//Get system parameter
uint32_t valWdogRd;
//SystemParam - WDOG
res = trio_GetSystemParameter_WDOG(&context, &valWdogRd);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_GetSystemParameter_WDOG returned error: %llx\n", res);
	ok = false;
}
if (valWdogRd != wdog)
{
	printf("trio_GetSystemParameter_WDOG returned different value: %d != %d\n",
valWdogRd, wdog);
	ok = false;
}
```

3.5. Digital IO operations.

```
//Set the digital output of channel 0.
res = trio_SetDOut2(&context, 0, true);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_SetDOut2 returned error: %llx\n", res);
	ok = false;
}

//Get the digital output of channel 0.
bool doutRd;
res = trio_GetDOut2(&context, 0, &doutRd);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_GetDout2 returned error: %llx\n", res);
	ok = false;
}
if (!doutRd)
{
	printf("trio_GetDout2 returned different value: %d != 1\n", doutRd);
	ok = false;
}

//Get the digital input of channel 0.
bool dinRd;
res = trio_GetDIn2(&context, 0, &dinRd);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_GetDIn2 returned error: %llx\n", res);
	ok = false;
}
```

3.6. Analog IO operations.

```
//Set the analog output of channel 0.
int aout = 100;
res = trio_SetAOut(&context, 0, aout);
if (res != trio_ErrorCode_NoError)
{
```

```
	printf("trio_SetAOut returned error: %llx\n", res);
	ok = false;
}

//Get the analog output of channel 0.
int aoutRd;
res = trio_GetAOut(&context, 0, &aoutRd);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_GetAOut returned error: %llx\n", res);
	ok = false;
}
if (aoutRd != aout)
{
	printf("trio_GetAOut returned different value: %d != 1\n", aoutRd);
	ok = false;
}

//Get the analog input of channel 0.
int ainRd;
res = trio_GetAIn(&context, 0, &ainRd);
if (res != trio_ErrorCode_NoError)
{
	printf("trio_GetAIn returned error: %llx\n", res);
	ok = false;
}
```

3.7. Motion commands.

```
//Single axis move
double pos = 1000;
res = trio_MoveRel1(&context, pos, axis); // Axis(0) moves the distance //1,000(
                                            relative position)
if (res != trio_ErrorCode_NoError)
{
	printf("trio_MoveRel1 returned error: %llx\n", res);
	ok = false;
}

//Two axes linear interpolation motion( relative position)
double positions[] = { 1000, 2000 };
uint32_t axes[] = { 0, 2 };
res = trio_SetMultiAxisBase(&context, axes, 2); // Set the base axes.
if (res != trio_ErrorCode_NoError)
{
	printf("trio_SetMultiAxisBase returned error: %llx\n", res);
	ok = false;
}
res = trio_MoveRel2(&context, positions,2, axis); // Axis(0) moves the distance
                                                    //1,000, while axis(2) moves 2,000
if (res != trio_ErrorCode_NoError)
{
	printf("trio_MoveRel2 returned error: %llx\n", res);
	ok = false;
}

//Sets continuous forward movement.
int mode = 2;
```

```
res = trio_Cancel2(&context, mode, axis); // Cancels all active and buffered moves
if (res != trio_ErrorCode_NoError)
{
	printf("trio_Cancel2 returned error: %llx\n", res);
	ok = false;
}
res = trio_Forward(&context, axis); // Axis(0) moves foward
if (res != trio_ErrorCode_NoError)
{
	printf("trio_Forward returned error: %llx\n", res);
	ok = false;
}
sleep_ms(3000); // Waiting for 3 seconds.
mode = 0;
res = trio_Cancel2(&context, mode, axis); // Cancel the currently executing move
if (res != trio_ErrorCode_NoError)
{
	printf("trio_Cancel2 returned error: %llx\n", res);
	ok = false;
}

//Sets continuous reverse movement.
res = trio_Reverse(&context, axis); // Axis(0) moves foward
if (res != trio_ErrorCode_NoError)
{
	printf("trio_Reverse returned error: %llx\n", res);
	ok = false;
}
sleep_ms(3000); // Waiting for 3 seconds.
res = trio_Cancel2(&context, mode, axis); // Cancel the currently executing move
if (res != trio_ErrorCode_NoError)
{
	printf("trio_Cancel2 returned error: %llx\n", res);
	ok = false;
}

//Two axes circular interpolation motion
res = trio_SetMultiAxisBase(&context, axes, 2); // Set the base axes.
if (res != trio_ErrorCode_NoError)
{
	printf("trio_SetMultiAxisBase returned error: %llx\n", res);
	ok = false;
}
res = trio_MoveCirc3(&context, 0, 2000, 0, 1000, 0,  1); /* Semicircular
trajectory(axis0 and axis2, clockwise direction), finish point (2000,0), centre
point (1000,0) */
if (res != trio_ErrorCode_NoError)
{
	printf("trio_MoveCirc3 returned error: %llx\n", res);
	ok = false;
}
sleep_ms(5000); // waiting for 5 seconds
```

3.8. Disconnect from the controller.

```
res = trio_CloseConnection(&context);
if (res != trio_ErrorCode_NoError)
{
```

```
        printf("trio_CloseConnection returned error: %llx\n", res);
        ok = false;
}
```

3.9. Destroy connection context.

```
// Destroy connection context
res = trio_DestroyContext(&context);
if (res != trio_ErrorCode_NoError)
{
        printf("trio_DestroyContext returned error: %llx\n", res);
        ok = false;
}
```