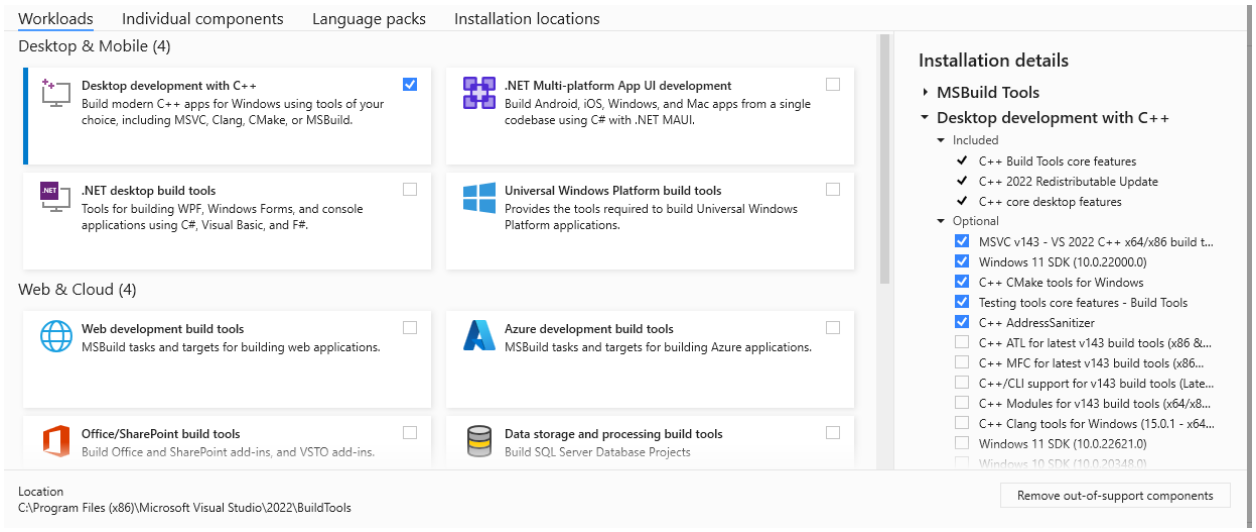
Build libmuParser dll for Windows 10 & 11

1. Download and Install VS Code (if not already installed): Download
2. Launch VS Code and Install the C/C++ plugins
    a. C/C++ by Microsoft (ms-vscode.cpptools)
    b. C/C++ Extension Pack by Microsoft (ms-vscode.cpptools-extension-pack)
3. Download and install the Build Tools for VIsual Studio 2022 (v17.4.3): Download
    a. Only the Desktop Development with C++ workload needs to be installed. This includes MSVC and CMake.



4. Download a copy of the libmuParser source code and extract it to a folder: https://github.com/beltoforion/muparser/releases
5. Open the folder in VS Code
6. Allow the cmake to configure the project
7. Set the build variant to Release
8. Select the build kit:
    a. Use “Visual Studio Build Tools Release 2022 - amd64” to build a 64-bit dll
    b. Use “Visual Studio Build Tools Release 2022 - x86” to build a 32-bit dll
9. Build
10. The dll will be output to the build subfolder if successful

Build libmuParser for Ubuntu 20.04

1. Install VS Code (if not already installed). Can be done through the Ubuntu Software manager.
2. Launch VS Code and Install the C/C++ plugins
    - C/C++ by Microsoft (ms-vscode.cpptools)
    - C/C++ Extension Pack by Microsoft (ms-vscode.cpptools-extension-pack)
3. Install the GCC, G++, and CMAKE

```
sudo apt update
sudo apt install build-essential cmake ninja-build
```

4. (optional) Install gcc-multilib if you want to build for an x86 target

```
sudo apt install gcc-multilib g++-multilib
```

5. Download a copy of the libmuparser source code and extract it to a folder: https://github.com/beltoforion/muparser/releases
6. Open the folder in VS Code
7. Allow the cmake to configure the project
8. Set the build variant to Release
9. Select the build kit:
    - GCC 9.4.0 x86_64-linux-gnu
10. (Optional) If you want to build for an x86 target, you need to modify the project's CMakeLists.txt file:
    - Change:

```
option(ENABLE_SAMPLES "Build the samples" ON)
```

      To:

```
option(ENABLE_SAMPLES "Build the samples" OFF)
```

    - Change:

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wno-long-long -pedantic")
```

      To:

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -m32 -Wall -Wno-long-long -pedantic")
```

11. Build
12. The shared object file will be output to the build subfolder if successful

Build libmuParser for cRIO RT Linux

1. In addition to the Windows installation of VSCode (above), follow these instructions to set up VSCode for cross compiling for a RT Linux target: https://nilrt-docs.ni.com/cross_compile/config_dev_system.html

2. Download the GNU C & C++ Compile Tools x64 from NI. I have used version 2018-2019.

   https://www.ni.com/en/support/downloads/software-products/download.gnu-c---c---compile-tools-x64.html#338442

3. Install the toolchain as described: https://nilrt-docs.ni.com/cross_compile/config_dev_system.html#installing-the-c-c-cross-compile-toolchains
4. CMAKE was already installed (above) so you can skip that part
5. Install ninja and add it to the system path environment variable as described:

   https://nilrt-docs.ni.com/cross_compile/config_dev_system.html#ninja

6. In a command prompt, navigate to the toolchain folder and run the environment-setup-core2-64-nilrt-linux.bat script.
7. Launch VS Code from the same command prompt, to ensure that the toolchain environment is set up correctly.
8. Download a copy of the libmuparser source code and extract it to a folder: https://github.com/beltoforion/muparser/releases
9. Open the folder in VS Code
10. Configuring the C/C++ Extension for IntelliSense:

    https://nilrt-docs.ni.com/cross_compile/config_vs_code.html#configuring-the-c-c-extension-for-intellisense

    You should end up with a c_cpp_properties.json file containing:

```
{
    "env": {
      "compilerSysroots": "C:/build/18.0/x64/sysroots/"
    },
    "configurations": [
      {
        "name": "NI Linux Real-Time x64",
        "compilerPath":
"${compilerSysroots}/i686-nilrtsdk-mingw32/usr/bin/x86_64-nilrt-linux/x86_64-nilrt-linux-gcc.exe",
        "compilerArgs": [
          "--sysroot=${compilerSysroots}/core2-64-nilrt-linux/"
        ],
        "includePath": [
          "${workspaceFolder}/",
          "${compilerSysroots}/core2-64-nilrt-linux/usr/include/"
        ],
        "intelliSenseMode": "gcc-x64"
      }
    ],
    "version": 4
```

```
  }
```

11. Select “NI Linux Real-Time x64” as the build kit
12. Modify the cMakeLists.txt file:

    Set build samples and use openMP to off

```
option(ENABLE_SAMPLES "Build the samples" OFF)
option(ENABLE_OPENMP "Enable OpenMP for multithreading" OFF)
```

    change the library name from muparser to muparser-lv

```
add_library(muparser-lv ${MUPARSER_SOURCES})
```

    **Requires changing subsequent library name references to muparser-lv. There are quite a few.

13. Build
14. The shared object file will be output to the build subfolder if successful

Modifications made to libmuParser distributed with LV-muParser

muParserDLL.cpp

1. Added callback functions

```
//---------------------------------------------------------------------------
// Callbacks for infix operators
muFloat_t Not(muFloat_t v) { return v == 0; }

muFloat_t BinNot(muFloat_t v) { return ~(uint32_t)v; }

//---------------------------------------------------------------------------
// Callbacks for Operators
static muFloat_t Mod(muFloat_t v1, muFloat_t v2) { return (muFloat_t)(fmod(v1,v2));}
static muFloat_t BinAnd(muFloat_t v1, muFloat_t v2) { return (muFloat_t)((uint32_t)v1 & (uint32_t)v2); }
static muFloat_t BinOr(muFloat_t v1, muFloat_t v2) { return (muFloat_t)((uint32_t)v1 | (uint32_t)v2); }
static muFloat_t BinXOr(muFloat_t v1, muFloat_t v2) { return (muFloat_t)((uint32_t)v1 ^ (uint32_t)v2); }
static muFloat_t BinShiftLeft(muFloat_t v1, muFloat_t v2) { return (muFloat_t)(0xFFFFFFFF & (uint32_t)v1 <<
(uint32_t)v2); }
static muFloat_t BinShiftRight(muFloat_t v1, muFloat_t v2) { return (muFloat_t)(0xFFFFFFFF & (uint32_t)v1 >>
(uint32_t)v2); }

//---------------------------------------------------------------------------
// Callbacks for value type recogniction


static int IsHexValue(const mu::char_type* a_szExpr, int* a_iPos, mu::value_type* a_fVal)   //Hex values must start
with "0x"
{
    if (a_szExpr[1] == 0 || (a_szExpr[0] != '0' || a_szExpr[1] != 'x'))
        return 0;

    unsigned iVal(0);

    // New code based on streams for UNICODE compliance:
    mu::stringstream_type::pos_type nPos(0);
    mu::stringstream_type ss(a_szExpr + 2);
    ss >> std::hex >> iVal;
    nPos = ss.tellg();

    if (nPos == (mu::stringstream_type::pos_type)0)
        return 1;

    *a_iPos += (int)(2 + nPos);
    *a_fVal = (mu::value_type)iVal;

    return 1;
}

static int IsBinValue(const mu::char_type* a_szExpr, int* a_iPos, mu::value_type* a_fVal)  //Bin values must start
with "0b"
{
    if (a_szExpr[1] == 0 || (a_szExpr[0] != '0' || a_szExpr[1] != 'b'))
        return 0;

    unsigned iVal = 0;
    unsigned iBits = sizeof(iVal) * 8;
    unsigned i = 0;
```

```
    for (i = 0; (a_szExpr[i + 2] == '0' || a_szExpr[i + 2] == '1') && i < iBits; ++i)
        iVal |= (int)(a_szExpr[i + 2] == '1') << ((iBits - 1) - i);

    if (i == 0)
        return 0;

    if (i == iBits)
        throw mu::Parser::exception_type(_T("Binary to integer conversion error (overflow)."));

    *a_fVal = (unsigned)(iVal >> (iBits - i));
    *a_iPos += i + 2;

    return 1;
}
//---------------------------------------------------------------------------
```

2. Modified mupCreate function to add the new functions:

```
API_EXPORT(muParserHandle_t) mupCreate(int nBaseType)
{
    muParserHandle_t handle;
    switch (nBaseType)
    {
    case  muBASETYPE_FLOAT:
        handle = (void*)(new ParserTag(muBASETYPE_FLOAT));
        mupDefineNameChars(handle, _T("0123456789_:abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"));
        mupDefineInfixOprt(handle, _T("!"), Not, mu::prINFIX, 0);
        mupDefineInfixOprt(handle, _T("~"), BinNot, mu::prINFIX, 0);
        mupDefineOprt(handle, _T("%"), Mod, mu::prMUL_DIV, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("&"), BinAnd, mu::prBAND, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("|"), BinOr, mu::prBOR, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("|^"), BinXOr, mu::prBOR, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("<<"), BinShiftLeft, mu::prCMP, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T(">>"), BinShiftRight, mu::prCMP, mu::oaLEFT, 0);

        mupAddValIdent(handle, IsHexValue);   //add support for hex-constants "0x123"
        mupAddValIdent(handle, IsBinValue);   //add support for bin-constants "0b1001"

        return handle;
    case  muBASETYPE_INT:
        handle = (void*)(new ParserTag(muBASETYPE_INT));
        mupDefineNameChars(handle, _T("0123456789_:abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"));
        mupDefineInfixOprt(handle, _T("!"), Not, mu::prINFIX, 0);
        mupDefineInfixOprt(handle, _T("~"), BinNot, mu::prINFIX, 0);
        mupDefineOprt(handle, _T("%"), Mod, mu::prMUL_DIV, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("&"), BinAnd, mu::prBAND, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("|"), BinOr, mu::prBOR, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("|^"), BinXOr, mu::prBOR, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T("<<"), BinShiftLeft, mu::prCMP, mu::oaLEFT, 0);
        mupDefineOprt(handle, _T(">>"), BinShiftRight, mu::prCMP, mu::oaLEFT, 0);

        mupAddValIdent(handle, IsHexValue);   //add support for hex-constants "0x123"
        mupAddValIdent(handle, IsBinValue);   //add support for bin-constants "0b1001"

        return handle;
    default:                  return nullptr;
    }
}
```