QT with Visual Studio 2008 (updated for Qt 4.6.1)

These directions will allow Qt4 to be installed and used by Visual Studio 2008.

Contents

1. Installation:

1. Apply service pack 1 to Visual Studio 2008 if it has not already been done.
2. Download the latest version of QT's source code (not the minGW binaries)
3. Extract the source into c:\Qt\4.x.x or a similar path. This path will be referred to as QTDIR for the remainder of this document. Qt compilation will fail if there are spaces in the path or if the path is changed after the compilation process has completed.
4. Create the environment variables:
   1. Go to Control Panel -> System -> Advanced -> Environment Variables
   2. Within the user variables section, click New to bring up the New User Variable dialog. User variables should be used (instead of system-wide environment variables) because multiple versions and configurations of Qt can be compiled and available on one computer simultaneously. By using User Variables, changes to one user's variables will not affect other users of the computer.
   3. Fill in the dialog. Fill in the name field with "QTDIR" and the value field with the path to the Qt source code.
   4. Click ok.
   5. Create a user environment variable PATH (or edit it if it already exists). Set the value to %QTDIR%\bin;%PATH% if PATH didn't already exist and prepend %QTDIR%\bin; if it did.
   6. Click ok twice.

In order to install QT 4.6.1, it is best to enable the Dx86 environment from within the Visual Studio Command Prompt. In order to complete these steps, the DirectX SDK must be installed. The latest DirectX SDK, as of this writing, is the February 2010 build (download). You can check for a newer version here.

Enabling the Dx86 environment:

1. Once the DirectX SDK is installed, navigate to: Start -> All Programs -> Microsoft DirectX SDK (February 2010) and right click on DirectX SDK Command Prompt.
2. Click on Properties
3. In the target field of the DirectX SDK Command Prompt Properties window, copy the string contained within the target field. If you are using the March 2009 build, the string reads as follows: "C:\Program

Files\Microsoft DirectX SDK (February 2010)\Utilities\Bin\dx_setenv.cmd"
   - *Note: the string will be in two sets of quotation marks. ONLY copy the string and one set of quotation marks. What you have copied should resemble the string shown in the step above.*
4. Open the Visual Studio 2008 Command Prompt by clicking: Start -> All Programs -> Microsoft Visual Studio 2008 -> Visual Studio Tools -> Visual Studio 2008 Command Prompt.
5. Paste the string you copied in step 3 into the Command prompt. You should receive the following message: "Dx86 target environment is now enabled. Dx86 host environment is now enabled."

1. Open the Visual Studio 2008 Command Prompt by clicking: Start -> All Programs -> Microsoft Visual Studio 2008 -> Visual Studio Tools -> Visual Studio 2008 Command Prompt.
2. Navigate to the Qt directory:

```
cd %QTDIR%
```

1. Configure Qt.

```
configure -debug-and-release -platform win32-msvc2008
```

help

Print a list of configuration options.

debug-and-release

Enable generation of both debug and release mode executables.

platform win32-msvc2008

Specifies the desired compiler.

1. . Choose which license you would like to use and enter y when prompted. The configuration process will take a few minutes.
2. Build Qt. This will take a long time.

```
nmake
```

1. Change to the demos directory.

```
cd %QTDIR%/demos
```

1. Delete unnecessary files from the demos directory.

```
nmake clean
```

1. Change to the examples directory.

```
cd %QTDIR%/examples
```

1. Delete unnecessary files from the examples directory.

```
nmake clean
```

1. Improve Debugger support for Qt datatypes.

Add the following lines to the file <Visual Studio Directory>/Common7/Packages/Debugger/autoexp.dat just before the line "[Visualizer]". Note: If you are on the Windows Vista or Windows 7 platform, you must open the text editor in "Run as Administrator" mode.

```
; Qt Integration QObject =classname=<staticMetaObject.d.stringdata,s>
superclassname=<staticMetaObject.d.superdata->d.stringdata,s>
QList<*>=size=<d->end,i> QLinkedList<*>=size=<d->end,i> QString=<d->data,su>
size=<d->size,u> QByteArray=<d->data,s> size=<d->size,u> QUrl
=<d->encodedOriginal.d->data,s> QUrlInfo =<d->name.d->data,su> QPoint =x=<xp>
y=<yp> QPointF =x=<xp> y=<yp> QRect =x1=<x1> y1=<y1> x2=<x2> y2=<y2> QRectF
=x=<xp> y=<yp> w=<w> h=<h> QSize =width=<wd> height=<ht> QSizeF =width=<wd>
height=<ht> QMap<*> =size=<d->size> QVector<*> =size=<d->size> QHash<*>
=size=<d->size> QVarLengthArray<*> =size=<s> data=<ptr> QFont
=family=<d->request.family.d->data,su> size=<d->request.pointSize, f> QDomNode
=name=<impl->name.d->data,su> value=<impl->value.d->data,su>
```

Add the following just after the [Visualizer] tag

```
QString{ preview ([$e.d->data,su]) stringview ([$e.d->data,sub]) children ( #(
d: $c.d, [size]: $c.d->size, [referenced]: $c.d->ref._q_value ) ) } QFileInfo{
preview ( #( "private=", $c.d_ptr ) ) }
;------------------------------------------------------------------------------
; QStringList
;------------------------------------------------------------------------------
QStringList{ preview ( #if (($c.d->end - $c.d->begin) <= 10) ( #( "[", $c.d->end
- $c.d->begin, "](", #array ( expr: (QString)(($c.d->array + $c.d->begin)[$i]),
size: $c.d->end-$c.d->begin ), ")" ) ) #else ( #( "[", $c.d->end - $c.d->begin,
"](", #array ( expr: (QString)(($c.d->array + $c.d->begin)[$i]), size: 10 ), ",
...)" ) ) ) }
;------------------------------------------------------------------------------
; QList, QQueue
;------------------------------------------------------------------------------
QList<*>|QQueue<*>{ preview ( #if ((sizeof($T1)) <= (sizeof(void *))) ( #if
(($c.d->end - $c.d->begin) <= 10) ( #( "[", $e.d->end - $e.d->begin , "](",
#array ( expr: ($T1)(($c.d->array + $c.d->begin)[$i]), size:
$c.d->end-$c.d->begin ), ")" ) ) #else ( #( "[", $e.d->end - $e.d->begin , "](",
#array ( expr: ($T1)(($c.d->array + $c.d->begin)[$i]), size: 10 ), ", ...)" ) )
) #else ( #( "[", $e.d->end - $e.d->begin , "](", #array ( expr:
*($T1*)(($c.d->array + $c.d->begin)[$i]), size: $c.d->end-$c.d->begin ), ")" ) )
) children ( #if ((sizeof($T1)) <= (sizeof(void *))) ( #array ( expr:
($T1)(($c.d->array + $c.d->begin)[$i]), size: $c.d->end-$c.d->begin ) ) #else (
#array ( expr: *($T1*)(($c.d->array + $c.d->begin)[$i]), size:
$c.d->end-$c.d->begin ) ) ) }
```

```
;------------------------------------------------------------------------------
; QList::iterator, QQueue::iterator
;------------------------------------------------------------------------------
```

```
QList<*>::iterator|QList<*>::const_iterator|QQueue<*>::iterator|QQueue<*>::const_itera
preview ( #( ($T1)$c.i->v ) ) children ( #( ptr: ($T1)$c.i->v ) ) }
;------------------------------------------------------------------------------
; QListIterator
;------------------------------------------------------------------------------
QListIterator<*>|QMutableListIterator<*> { preview ( #( ($T1)$c.i.i->v ) )
children ( #( ptr: ($T1)$c.i.i->v ) ) }
;------------------------------------------------------------------------------
; QLinkedList
;------------------------------------------------------------------------------
QLinkedList<*>{ preview ( #if ($e.d->size >= 10) ( #( "[", $e.d->size, "](",
#list ( head: $c.d->n, size: 10, next: n ) : ( (*(QLinkedListNode<$T1>*)&$e).t
), ", ...)" ) ) #else ( #( "[", $e.d->size, "](", #list ( head: $c.d->n, size:
$c.d->size, next: n ) : ( (*(QLinkedListNode<$T1>*)&$e).t ), ")" ) ) ) children
( #( #list ( head: $c.d->n, size: $c.d->size, next: n ) :
(*(QLinkedListNode<$T1>*)&$e).t ) ) }
;------------------------------------------------------------------------------
; QLinkedList::iterator
;------------------------------------------------------------------------------
QLinkedList<*>::iterator|QLinkedList<*>::const_iterator{ preview ( #( $e.i->t )
) children ( #( ptr: $e.i->t ) ) }
;------------------------------------------------------------------------------
; QVector, QStack
;------------------------------------------------------------------------------
QVector<*>|QStack<*>{ preview ( #if ($c.d->size <= 10) ( #( "[", $c.d->size,
"](", #array ( expr: $c.d->array[$i], size: $c.d->size ), ")" ) ) #else ( #(
"[", $c.d->size, "](", #array ( expr: $c.d->array[$i], size: 10 ), ", ...)" ) )
) children ( #array ( expr: $c.d->array[$i], size: $c.d->size ) ) }
;------------------------------------------------------------------------------
; QMap::Node
;------------------------------------------------------------------------------
QMap<*,*>::Node{ preview ( #( "(", $c.key,"; ", $c.value, ")" ) ) children ( #(
key: $c.key, value: $c.value ) ) }
;------------------------------------------------------------------------------
; QMap
;------------------------------------------------------------------------------
QMap<*>{ preview ( #( "[", $e.d->size, "](", #tree ( head: $c.d->forward, size:
$c.d->size, left: backward, right: forward ) : $e, ")" ) ) children ( #tree (
head: $c.d->forward[0], skip: $c.d, size : $c.d->size, left : backward, right :
forward ) : ( (QMap<$T1>::Node*)((char*)&$e -
(sizeof(*(QMap<$T1>::PayloadNode*)0) - sizeof(QMapData::Node*))) ) ) ) }
;------------------------------------------------------------------------------
; QMap::iterator
;------------------------------------------------------------------------------
QMap<*>::iterator|QMap<*>::const_iterator{ preview ( #(
(QMap<$T1>::Node*)((char*)$e.i - (sizeof(*(QMap<$T1>::PayloadNode*)0) -
sizeof(QMapData::Node*))) ) ) children ( #( ptr: (QMap<$T1>::Node*)((char*)$e.i
- (sizeof(*(QMap<$T1>::PayloadNode*)0) - sizeof(QMapData::Node*))) ) ) }
```

2. Creating a Qt project with Visual Studio 2008

Configure nmake. Makefiles require that nmake be properly configured for the new project. This can be accomplished in two locations, within the Makefile Project Wizard or within the project properties dialog after the project has been fully created. (TODO: Complete Intellisense section)

2.1 Configuring nmake with the Makefile Project Wizard.

- Ensure the environment variable %QTDIR% points to your installation of QT.
- Open Visual Studio and select File -> New ->Project.
- Select General and then Makefile Project.
- After providing a name and location for the new Makefile project, click Ok.

Fill in the following fields:
 Build Command Line:

```
 %QTDIR%\bin\qmake nmake debug
```

1. Select Debug Configuration Settings.
2.

Rebuild All Command Line:

```
%QTDIR%\bin\qmake nmake debug-clean nmake debug
```

Clean Command Line:

```
%QTDIR%\bin\qmake nmake debug-clean
```

Output:

```
debug\[ApplicationName].exe
```

Fill in the follwing fields:
 Build Command Line:

```
 %QTDIR%\bin\qmake nmake release
```

- Select Release Configuration Settings.
- Uncheck the Same as debug configuration box.
-

Rebuild All Command Line:

```
%QTDIR%\bin\qmake nmake release-clean nmake release
```

Clean Command Line:

```
%QTDIR%\bin\qmake nmake release-clean
```

Output:

```
release\[ApplicationName].exe
```

- Click Finish to complete the creation of the new Makefile project.

2.2 Configuring nmake with Properties Pages dialog

Fill in the four properties listed under General.
 Build Command Line:

```
 %QTDIR%\bin\qmake nmake debug
```

1. With the project open, right click on the name of the project within the Solution Explorer toolbar (generally on the left side of the IDE).
2. Select Properties in the popup menu.
3. Select NMake to access the nmake configuration section.
4. Under Configuration, select Debug to edit the Debug Configuration properties.
5.

Rebuild All Command Line:

```
%QTDIR%\bin\qmake nmake debug-clean nmake debug
```

Clean Command Line:

```
%QTDIR%\bin\qmake nmake debug-clean
```

Output:

```
debug\[ApplicationName].exe
```

Fill in the four properties listed under General.
 Build Command Line:

```
 %QTDIR%\bin\qmake nmake release
```

- Under configuration, select Release to edit the Release Configuration properties. If asked to save settings, select Yes.
-

Rebuild All Command Line:

```
%QTDIR%\bin\qmake nmake release-clean nmake release
```

Clean Command Line:

```
%QTDIR%\bin\qmake nmake release-clean
```

Output:

```
release\[ApplicationName].exe
```

- Click Ok to finish configuring nmake.

Depending on the desired type of project, copy the corresponding example into the .pro file (replacing ApplicationName with the application's name):
 GUI Application:

```
 TEMPLATE += app CONFIG += qt warn_on no_keywords embed_manifest_exe QT +=
TARGET = ApplicationName SOURCES = HEADERS = LIBS += # Treat warnings as errors
win32:QMAKE_CXXFLAGS += /WX CONFIG(debug, debug|release){ # Debug build options
# Enable a read-only console window (i.e. for printf etc.) # CONFIG += console }
else{ # Release build options # Enable a read-only console window (i.e. for
printf etc.) # CONFIG += console }
```

- Create a file with an extension of .pro
-

Console Application:

```
TEMPLATE += app CONFIG += qt warn_on no_keywords console embed_manifest_exe QT
+= TARGET = ApplicationName SOURCES = HEADERS = LIBS += # Treat warnings as
errors win32:QMAKE_CXXFLAGS += /WX CONFIG(debug, debug|release){ # Debug build
options } else{ # Release build options }
```

DLL:

```
TEMPLATE += lib CONFIG += qt warn_on no_keywords dll embed_manifest_dll QT +=
TARGET = ApplicationName SOURCES = HEADERS = LIBS += # Treat warnings as errors
win32:QMAKE_CXXFLAGS += /WX CONFIG(debug, debug|release){ # Debug build options
} else{ # Release build options }
```

Static Library:

```
TEMPLATE += lib CONFIG += qt warn_on no_keywords staticlib QT += TARGET =
ApplicationName SOURCES = HEADERS = LIBS += # Treat warnings as errors
win32:QMAKE_CXXFLAGS += /WX CONFIG(debug, debug|release){ # Debug build options
} else{ # Release build options }
```

Qt Plugin:

```
TEMPLATE += lib CONFIG += qt warn_on no_keywords plugin QT += TARGET =
ApplicationName SOURCES = HEADERS = LIBS += # Treat warnings as errors
win32:QMAKE_CXXFLAGS += /WX CONFIG(debug, debug|release){ # Debug build options
} else{ # Release build options }
```

If the Qt project will be linked against static libraries that will undergo development at the same time as the application, add the following to the .pro file:

```
CONFIG(debug, debug|release){ # Debug build options LIBS +=
path/to/debug/library/library.lib POST_TARGETDEPS += $$LIBS } else{ # Release
build options LIBS += path/to/release/library/library.lib POST_TARGETDEPS +=
$$LIBS }
```

- Create and fill in the application's source files. Each source and header file needs to be listed in the appropriate sections of the applications .pro file (HEADERS and SOURCES). Make sure to always updates this file when new header files and source files are added to the project. Additionally, this approach requires frequently rebuilding of the entire project to ensure that everything is up to date.

3. QtWizard

Because generating Qt makefile projects within Visual Studio can be tedious, especially when developing large numbers of smaller applications, a Visual Studio wizard has been produced that automates parts of the process. The installer for the wizard can be located at http://www.sanfordfreedman.com/personal/projects/. The QtWizard integrates into Visual Studio and lets the user generate a Qt makefile project in the same manner as a standard Win32 or other inherently supported project type.

4. IntelliSense

Add the following lines to the Visual Studio include file paths dialog.

In order to use V-Tune with release builds of QT applications, the following needs to be added to the .pro file.

- When working with Qt (and qmake), running the rebuild command is often required. This primarily becomes an issue when changes are made to the .pro file or when Q_OBJECT is added or removed from files.
- In order to not pollute the global namespace, the above .pro file examples use the no_keywords configuration option. This option disables Qt specific macros. Instead of using the standard Qt macros, use the Q_X equivalents (replacing X with the name of the macro, i.e. Q_SLOTS).
-

5. External References

Official Qt 4.6 Documentation
Official QT Website
Qt Centre - A Qt Community Wiki and forum
QtNode - A Qt Community Wiki