Clarion Handy Tools

NEWS - JUNE 2017

June 27, 2017
Programming Clarion For Touch and the UWP

UWP (Universal Windows Platform) and the evolution of touch development...

(Article reposted from Windows Developer, May 25, 2017)

A note from Gus before you read this article...

The lines between mouse-oriented desktop app interfaces and touch-oriented tablet/phone interfaces continue to blurr at an ever-faster rate.

With Clarion applications still heavily predjudiced towards mouse-oriented desktop-style interfaces (even though we now have finger-touch capability) it's becoming more and more obvious that Clarion development remains resistant to touch interfaces and touch-interactions with the application, primarily because the tools provided in the IDE and the Clarion run-time, lag quite a distance behind the mainstream and the UWP. (Though at CHT, we're working on that as you will see in upcoming articles.)

It's not so much that it can't be done from Clarion, but because our personal design paradigms and requisite skill-sets fall behind over time.

This article (below) struck our interest this morning as we opened the latest version of Visual Studio 2017, because it brings into focus some on-going work we're doing here at CHT creating more touch-friendly interfaces for Clarion applications. For example, menu-tiles and form designs for Clarion applications that derive from HTML/CSS instead of Clarion's static Screen Designer. These designs flow and flex within the "screen" or "window" frame regardless of aspect ratio, vertical or horizontal.

We'll be posting apps, articles and a few videos over the coming weeks and months that illustrate what can be done to make Clarion applications less desktop-app centric and how to design user-interfaces that continue to work despite the shape and size of the "screen".

We hope you'll be entertained and perhaps elightened. And also that you'll study and learn to interact with and apply, the concepts, tools and techniques we're providing.

Here, now, is the Windows Developer article: ***UPW and the Evolution of Touch Development...***

How is programming for touch development on the Universal Windows Platform (UWP) different from mouse and keyboard development in Windows Forms? This post will cover some of the subtle differences between the two and how to use the most advanced tools for building smooth touch experiences.

The ways we communicate with our computers have gone through many changes over the years. For a long time, interactions were governed by the keyboard.

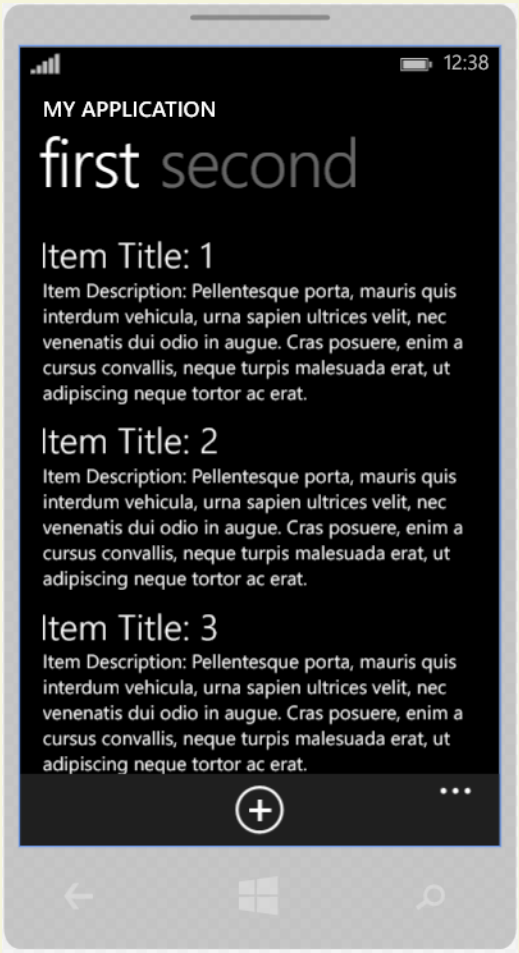
Then, the Graphical User Interface (GUI) came along, which not only introduced the mouse, but drastically altered how home screens and apps looked. The success of Windows 95 and Mac OS cemented that user experience, popularized personal computing and changed our computing landscape.

While Natural User Interfaces (NUI) have been around for a long time, the transition to touch interfaces has happened both more gradually and more quietly. In part, this is because it occurred hand-in-hand with the growth of smartphone-based mobile computing and we have a tendency not to see our phones as centers of computing power, but rather as accessories or appliances. In part, though, the smooth transition occurred because the creators of development tools such as Visual Studio made a conscious effort to protect developers from these changes by making touch and mouse (or tap and click) appear to be the same thing. As a consequence, we moved from a mouse-centric development world to touch-

friendly development world without really noticing the tectonic shift that occurred beneath our feet.

What does touch-friendly mean?

When touch-enabled tablets first arrived for Windows, you pretty much just used your finger as if it was a mouse to manipulate applications designed for the mouse. It was difficult because buttons tended to be too small and you couldn't see visual affordances, designed for mouse interactions, hidden beneath your fingers. Even if your device supported touch, the apps running on it continued to be mouse-centric. Similarly, early mobile devices came with a stylus because a stylus could manipulate those tiny buttons.



Phone apps mark the transition from mouse-centric to touch-friendly development. Phone apps weren't just touch-first, of course. They were touch-only. They forced changes in the design, layout and controls used in apps to make interaction easier for smartphone users. These in turn were eventually incorporated into the UWP platform. While the UWP platform supports both touch and mouse interactions, as well as a variety of other inputs, when you develop apps and controls for it, you should think of it first as a touch interface. Mouse interactions should be added on secondarily.

There are many overlaps between touch gestures and mouse interactions that make this easier. For instance, when you tap on a UWP button or click on it, both the Tapped event and the Click event are triggered.

Similarly, events such as ManipulationStarted and ManipulationCompleted do not differentiate between touch and mouse. Given the amount of effort that has gone into blurring the difference between touch and mouse in UWP, it is worth asking ourselves, when is a click not a click?

The age-old question: click or tapped

The Button control is an interesting island of GUI behavior in a touch-first world. If you double-click on a button in the Visual Studio designer, Visual Studio will wire up a Click event handler for you in XAML and in code-behind. This is there for backwards compatibility since, as pointed out above, touch and mouse interactions will both throw Click as well as Tapped events.

In order to develop in a touch-first way, however, you should handle the Tapped event rather than the Click. This becomes important when you need to distinguish Tapped from other touch events like DoubleTapped and Holding (the latter cannot even be emulated with a mouse).

```
<Button Content="My Button"
        Click="Button_Click"
        Tapped="Button_Tapped"
        RightTapped="Button_RightTapped"
        DoubleTapped="Button_DoubleTapped"
        Holding="Button_Holding"
        ManipulationMode="All"
        ManipulationStarted="Button_ManipulationStarted"
        ManipulationCompleted="Button_ManipulationCompleted"
/>
```

Once you have confirmed that your app works well for touch, you should add interactions for the mouse and other input vectors. For instance, you can capture clicks of the right mouse button by handling the oddly named RightTapped event.

```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="PointerOver"/>
        <VisualState x:Name="Pressed"/>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

Article continued here:

UPW And The Evolution Of Touch Development

Enjoy.

June 23, 2017
CHT Tile Menus Progress Report

We thought you might be interested in our "CHT Tile Menus" project to see where that might be headed at this early stage. *But first, let's review the purpose of this work.*

The "finger-on-screen" nature of Windows 10 in combination with less mouse-oriented navigation is proceeding away from fixed-interface designs at a pretty rapid clip. And the powerful data management strengths inherent in Clarion applications are perhaps not matched by Clarion's user-interface-creation capabilities.

We need a way to quickly and easily create, say a FRAME window that can act as a jump-into the procedures comprising your application, or to act as a launch-pad for applications in a suite of related applications. We need the frame to work equally well on a smallish tablet (or phone) as well as on a large-monitor desktop.

The early, experimental example we're presenting here is an app launch-pad. This does away with fixed-size controls and *fat-finger-unfriendly drop menus*. The screen is stretchy, so that the tiles that you see here are sized by percentages rather than fixed pixel sizes. When the screen gets to a certain point (tall and narrow, for instance) the tiles will begin to wrap and the screen goes into vertical scroll mode. So none of the tiles ever gets so small that they become too miniscule to use, even on a small tablet of phone.

The screen window is still a Clarion window, but the tiles are created and managed by HTML and CSS. So in this example, when you click on any tile, the Clarion window underneath sees that as an ordinary event (same as when you click on a button) and is able to launch an application or call a procedure which is exactly what you're already doing in ordinary Clarion.

For any of you who are allergic to (read: *scared shittless of*) HTML/CSS design, we are able to give you *Tile Window Templates* that operate on generated-CSS/HTML style sheets with different layout/design schemes so that creating a window of the sort in the next four images boils down to dropping a template or two, filling in some template prompts and generating your Clarion app as usual.

Here now, is one example "Frame" window using "stretchy" and "self-wrapping" tiles that will be available to you as an example in time for our early July 2017-3rd quarter build. In addition to this demo app,

which launches a variety of CHT example applications, there will also be a version of HNDPEOPLE_LBX.APP which launches the procedures now found in that application from a "Frame" consisting of tiles instead of menus or tool-bar buttons.

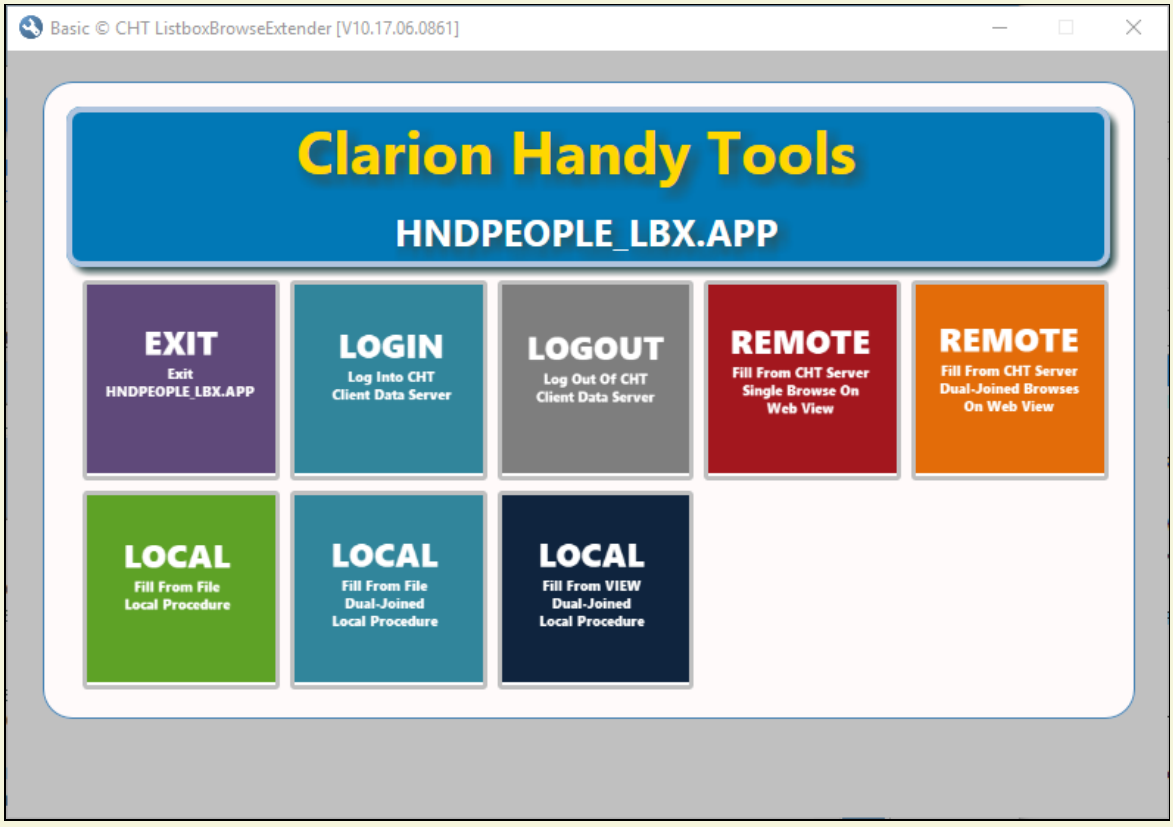
HNDPEOPLE_LBX.APP
Example Tile-Frame

We realized that once you create one of these tile frame procedures that all you have to do is import it to any other app, change the HTML file displaying in it and the CASE statement inside the procedure that executes the menus and it's going to run.

So here's a roughed-in version of the HNDPEOPLE_LBX.APP with the old FRAME or MAIN procedure removed and one of these CHT Tile-Menu procedures imported into it's place.

We want still, to add a bunch more tiles for some other functions such as changing the screen size and aspect ratio instantly to some pre-set sizes. But you get the idea.

The work involved was probably less than creating a new pure-Clarion procedure from scratch. Certainly no more than importing any standard Clarion MAIN procedure from another app and then modifying the menus. *Less work actually!*



Click any tile and the procedure or function that's described is executed.

This is a "stretchy" window for which you do not have to handle tile wrapping or shaping. That all happens automatically. I've downsized the image a bit for display here on the forum. The natural size of the window is as large or as small as the last time you resized it.

June 16, 2017
Build Update 21B.02.00 Now Available

As of 5:30 PM *Eastern Daylight Time* we completed posting CHT Build Update 21B.02.00.

We've download-tested and verified this latest update and it's ready for you to use any time you want it.

Run your *\accessory\hnd\hnd_wbupdaterC10.exe*, check "Auto-Continue" and then the "Continue" button and CHT's installation will update itself fully.

Not all installation containers were changed, obviously, so our installer/updater will only download and extract containers that were changed from your existing installation.

And did we mention you should always keep "Auto-Continue" checked? This makes the two phase installation of download and extraction into a one-phase operation by pushing the continue switch for you after you have pushed it initially to start.

Keep an eye on this *JUNE 2017 WHAT's NEW PAGE* and even our *JULY 2017 WHAT'S NEW PAGE* for information regarding this new build update, and changes or additions included.

June What's New (HTML)

June What's New (PDF)

June 12, 2017
New CHT Snap-In: CHTSNAP2PDF.EXE

We've sourced, in C#, a marvellous PDF library that amongst other things, can convert a well-formed HTML 5 document directly into an exact duplicate in PDF format.

For a few years we've been using the Windows 10 print driver to make this conversion from HTML 5 to PDF. And while this print driver works well, it does not enable the internal and external links in the HTML document.

Our new SNAP tool handles links brilliantly and we can build it right into our documentation apps as a CHT Snap-In. For example: *HNDTPXHTNEXT.APP, HNDAPPSPLASHTOHTML.APP, HNDCLXHT.APP* as well as a few others like, *HNDDOCUMENTBUILDER.APP, HNDPREVIEWER.APP, HNDPRINTHTML.APP*.

Check out a version of CHTTEMPLATES.HTML printed to pdf and try some of the internal links:

CHTTEMPLATES.PDF

2 Minute Video CHTTEMPLATES.PDF

June 8, 2017
Clarion Window Structures with Camtasia

We've been putting off making videos related to HNDPEOPLE_LBX.APP and the related LBX templates we're working on. So here's our back-story on that delay.

To make videos we use Camtasia, which while an awesome product, is a PITA to use when your app

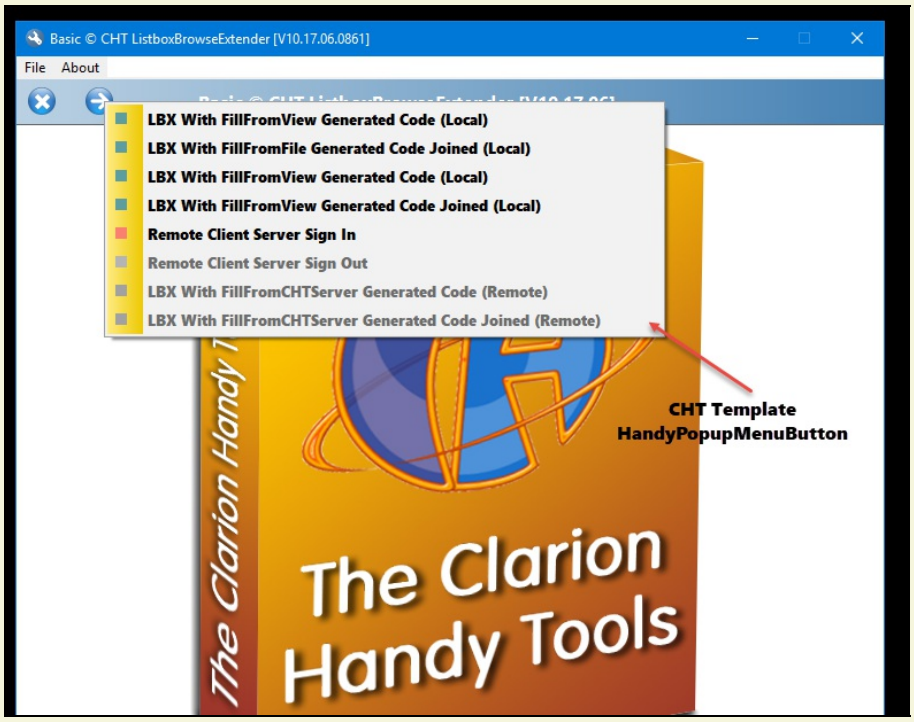
implements Clarion menus on a Clarion **Window** structure. *It doesn't happen on Clarion FRAME structures*.

Camtasia grabs certain call-back APIs on the window for itself when the recorder is running, and that basically kills Clarion menus. They just won't drop down.

To avoid the frustration when making videos with Camtasia, we added a CHT *HandyPopUpMenuButton* template on the toolbar of HNDPEOPLE_LBX.APP.

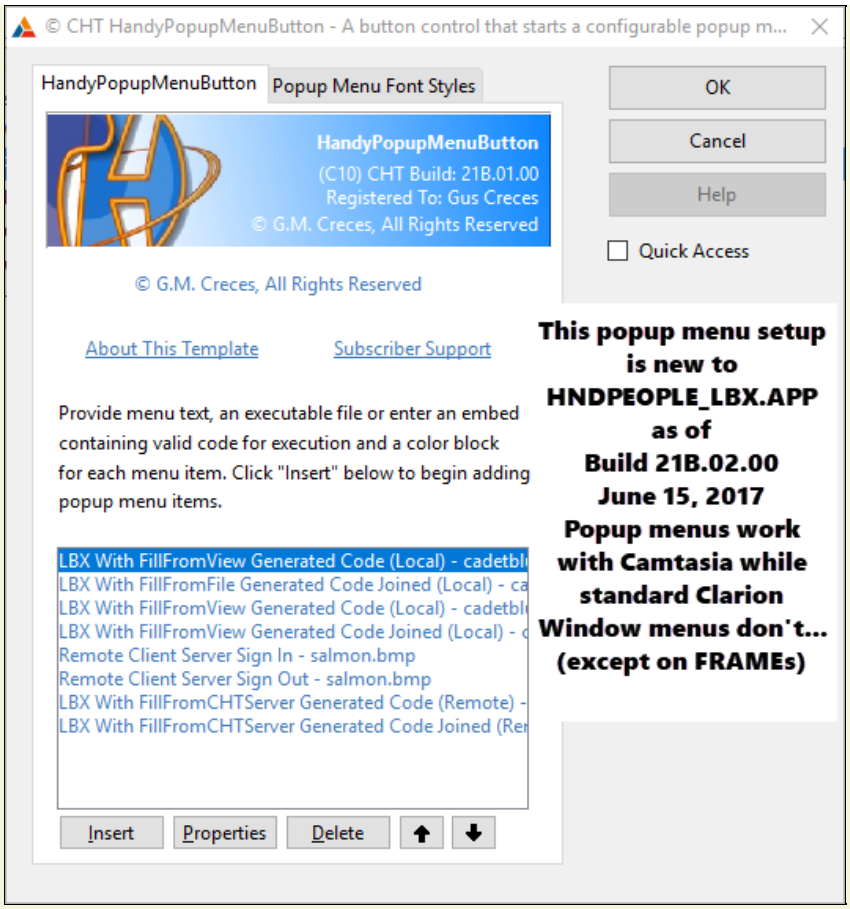
This button displays a popup menu when it is pushed/clicked. The template is configured to call the same six procedures being called from the HNDPEOPLE_LBX.APP standard menus. And this works just fine with Camtasia.

A CHT Popup Menu Example

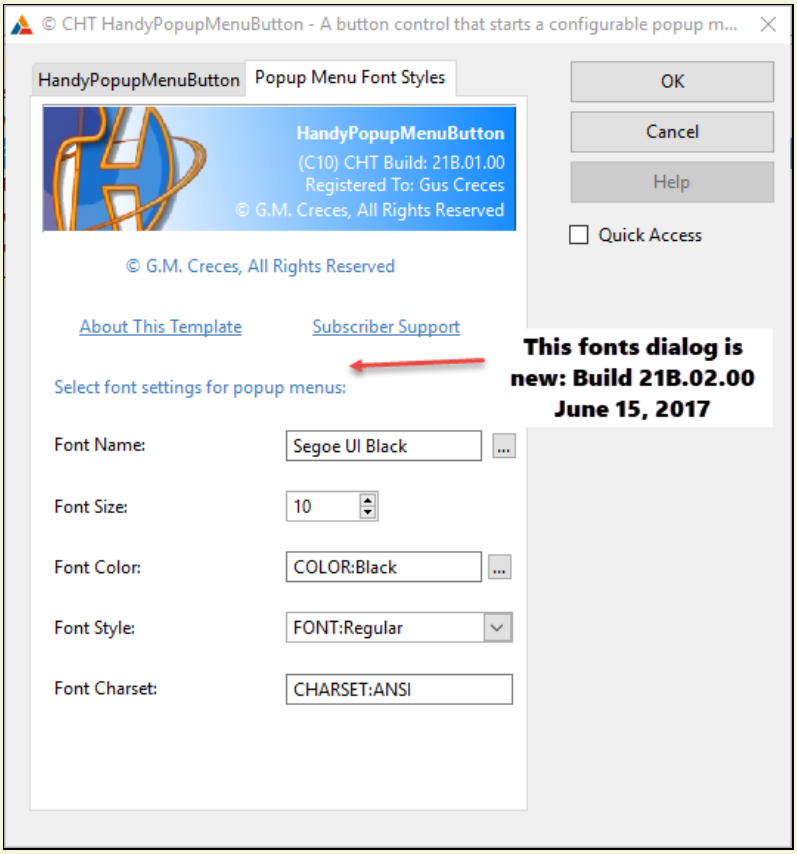


Latest CHT Popup Menu Enhancements

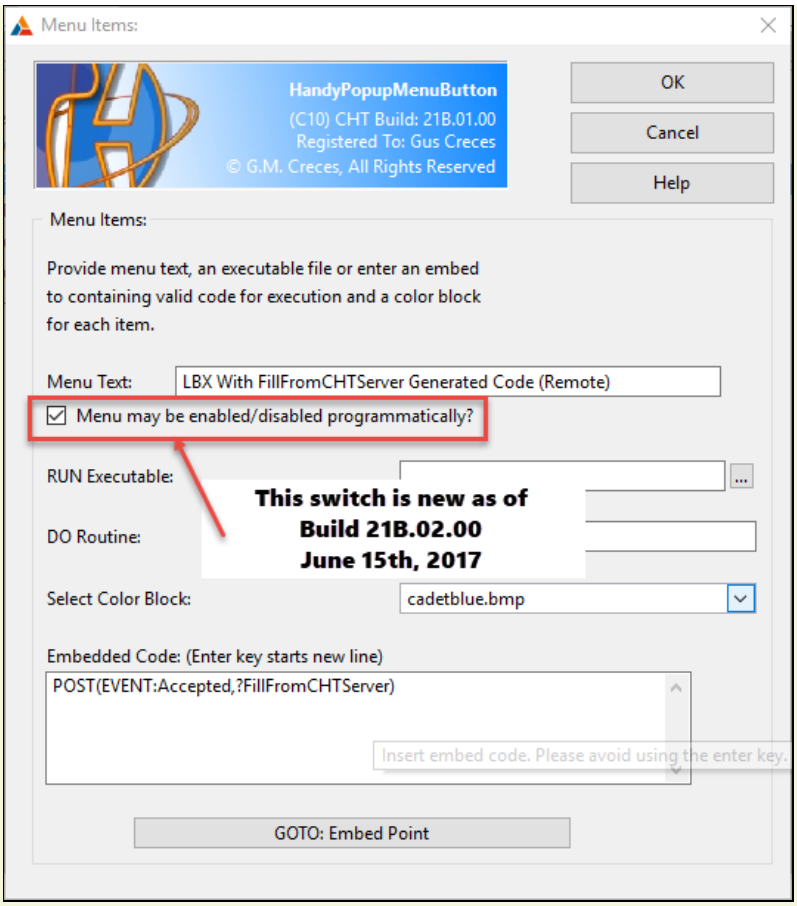
While we were at it we added a few features to the CHT HandyPopupMenuButton template: namely colorization, font control and enable/disable control, all of which you see being used in the screen snap above.



Configure Popup Menus Font Styles

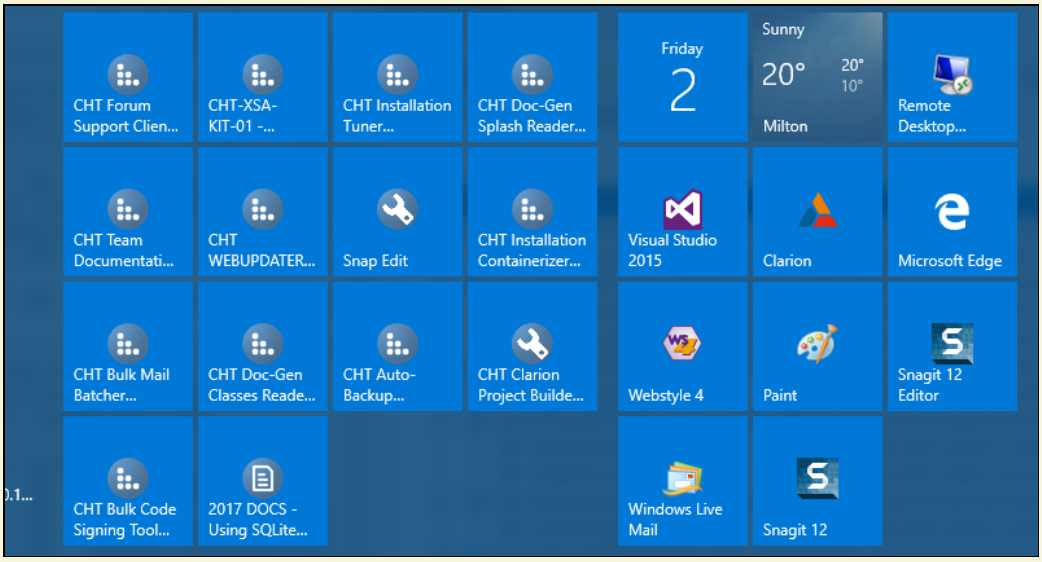


Programmatically Enable or Disable Menu Items



Tile Menus Template (Coming Soon)

In the touch-screen Windows 10 environment, it would be practical to have a template that creates a "Windows 10 Style" tiles-menu popup, that kind of emulates the windows tile screen we get on the Win10 start-menu. (see next image)



With fat fingers on a Windows 10 screen, tile menus are easier to use than windows standard menus and even CHT's popup menu. So we've begun work on creating this via a new CHT template and pop-tiles class.

This new feature likely won't appear until 4th quarter 2017, since we've lots of other work on our '*plates*, including the LBX videos we promised but this tiles menu is going to be a useful and utilitarian item for all of our apps.

If you have an opinion and/or a suggestion feel free to comment via the CHT Developer Forum.

June 1, 2017
The Latest Docs
(Regenerated Monthly)

The latest template documents (HTML/PDF) are here:

CHT Template Docs

(HTML)

(PDF)

The latest demo application docs are here:

CHT Application Docs

(HTML)

(PDF)

The latest utility application docs are here:

CHT Utility Applications Docs

(HTML)

(PDF)

The latest BATCH-BOT and SNAP-IN application docs are here:

CHT Batch-Bot Application Docs

(HTML)

(PDF)

The latest classes docs are here:

CHT Classes Docs

(HTML)

(PDF)