Subject: Re: [solved] ScatterCtrl: program access to the "Measures" ?
Posted by xrysf03 on Thu, 14 Nov 2019 11:51:55 GMT
View Forum Message <> Reply to Message

Dear Koldo and everyone,

i have one last snippet that I'd like to append to this topic, again for people coming after me:

The ScatterCtrl allows me to update the contents of the "data series buffer" after I have passed the buffer to the ScatterCtrl instance. The fixed-length buffer is passed by a pointer, and the documentation contains enough of a warning that the storage for the buffer must remain allocated until either the ScatterCtrl object is destroyed or the particular "data series" is withdrawn from the ScatterCtrl object using the RemoveSeries() or RemoveAllSeries() method. So: apparently ScatterCtrl merely "refers" to the external buffer, owned by me - and if I update some data in the buffer, I can tell the ScatterControl object to update its appearance using the Refresh() method.

In my overview spectrum scanner toy app, I'm using this to keep the user entertained: to show some data regularly as I obtain it from the radio. I scan the band in "chunks", corresponding to the "channel bandwidth" available from the radio (about 2 MSps = 2 MHz, centered on the tuned frequency). I split the broader band into 2MHz channels that I scan in sequence. And I can display each channel as soon as the sampling is done and the FFT gets calculated. To the user, the band-scan starts growing from the left to the right "in real time" - displayed by the ScatterCtrl object.

The following is a piece of pseudo-code, showing how I go about it:

```
// at the click of a button = in a "button event callback":
allocate global_buffer[buf_len]; // the aggregate buffer spanning all channels
zero_fill(global_buffer); // the chart starts out showing a flat line at y=0
Chart1.AddSeries(global_buffer, buf_len, starting_freq, channel_width);
Chart1.adjust_axis_units_and_view_dimensions();
Ctrl::ProcessEvents();

for (each channel)
{
 tune_into(channel);
 sample_chunk_from_channel();
 calc_FFT_for_the_channel();
 // project per channel FFT results into the overview buffer
 memcpy(global_buffer+chnl_offset, channel_FFT_output, chnl_data_len);
 Chart1.Refresh();    // tell the ScatterCtrl to update its output
 Ctrl::ProcessEvents();  // allow the GUI to actually carry out the updates pending :-)
}
```

And the key point why I'm writing this blurb is: Ctrl::ProcessEvents() :)
This turns out to be a key ingredient.

Why:

The GUI runs in a thread, handling Windows messages (those translate into U++ Events).
The message-driven programming model mostly ticks away swiftly, as the events are swiftly handled.
But: my band-scan activity is different. It hogs a long period of the single GUI thread's time.
Without Ctrl::ProcessEvents(), my app behaves this way:
While my lengthy "button click service routine" is running, taking its loops in scanning the channels,
the whole GUI is stuck, does not respond to events - that including the events I would like to incite by calling Chart1.Refresh().
The whole band scan can take a couple dozen seconds - during which time, nothing on the screen moves. Even my "gradual" updates of the data series in the chart (ScatterCtrl) all appear to rush in only at the very end of the band-scanning function, rather than gradually - in spite of me calling Chart1.Refresh() after every channel is processed.

Again the sprinkle of secret sauce here is called Ctrl::ProcessEvents(). If I call it after every Refresh(), the chart actually updates, the "go" button returns to its "unclicked" position, generally the GUI becomes more "alive" :) during the band-scan.

Still it needs to be said that the GUI does not actually get fully alive, as it's still not asynchronous to my band-scan function. To let the GUI respond to events in the usual way, while the band-scan is running, I'd have to run the scan in a background thread. Which has been my next goal anyway. Being conversant in Posix Threads and locking primitives, I can't wait to learn how much of those goodies has U++ possibly managed to include in its multi-platform API :) Last night I've discovered a nice U++ forum thread on the topic.

I hope this helps someone...

Frank

P.S.: For people who remember Delphi, ProcessMessages() might ring a bell :)