Best Practices for Igor Pro

Thomas Braun <thomas.braun@byte-physics.de>

4.5.2020
Version: 0.1.1

1 General hints

- Learn how to use wave assignment statements using `p, q, r, s`
- Avoid GUI functions like `ControlInfo` in performance critical code as they are comparatively slow
- Use builtins instead of homebrown algorithms, operations like `MatrixOP` and `Extract` do their job fast and good
- Keep the number of memory allocations low, the most common operations to look out for are `Make`, `Redimension` and `Duplicate`

2 Anti Patterns

2.1 if and do/while instead of for loops

Bad:

```
if(i > 0)
    do
        // do stuff
    while(i < count)
endif
```

Good:

```
for(i = 0; i < count; i += 1)
    // do stuff
endfor
```

Reasons:

- Easier to understand

- Shorter
- Less indentation

2.2 Inconsistent WAVE/DFREF/NVAR/SVAR usage

Bad:

```
WAVE/Z wv = mywave
numRows = DimSize(wv, 0)
```

Good:

```
// either if the wave always exists, don't add /Z
WAVE wv = mywave
numRows = DimSize(wv, 0)

// or, if it might not exist, handle that case
WAVE/Z wv = mywave

if(!WaveExists(wv))
    Abort "Missing wave"
endif

numRows = DimSize(wv, 0)
```

Reasons:

- Unclear intent
- Possibly incorrect

2.3 Treating boolean variables as non-boolean

Bad:

```
// code which shows that var can be either 1 or 0
if(var == 1)
    // code
elseif(var == 0)
    // code
endif
```

Good:

```
// In case var can be only 1 or 0, write it as
if(var)
    // code
```

```
else
    // code
endif

// Or handle the case where var != 1 and var != 0
if(var == 1)
    // code
elseif(var == 0)
    // code
else
    // new code
endif
```

Reasons:

- Unclear intent
- Possibly incorrect

2.4 Superfluous comparison for boolean values

Bad:

```
if(WaveExists(wv) == 1)
    // code
endif
```

Good:

```
if(WaveExists(wv))
    // code
endif
```

Reasons:

- Less verbose code

2.5 Unnecessary variables

Bad:

```
variable var = GetVariable()
SetVariable control value= _NUM:var
// code which does not use var anymore
```

Good:

```
// Some reports indicate that this rewrite is not always possible.
// Please report all such cases to your local Igor Pro guru
// or to WaveMetrics directly
SetVariable control value= _NUM:GetVariable()
```

Reasons:

- Shorter code
- Less variables

2.6 Wave/Datafolder creator functions don't return the just created objects

Bad:

```
Function CreateWave(param)
    variable param

    Make myWave
End

Function someOtherFunction()

    WAVE/Z myWave

    if(!WaveExists(myWave))
        CreateWave(param)
    endif

    WAVE myWave
End
```

Good:

```
Function/Wave GetWave(param)
    variable param

    // GetWaveFolder has the same logic as GetWave
    // Creates the datafolder if it does not exist
    // and returns a reference to it
    DFREF dfr = GetWaveFolder(param)

    WAVE/Z/SDFR=dfr data
    if(WaveExists(data))
        return data
    endif

    Make dfr:data/Wave=data
    // fill wave with default content

    return data
```

```
End

Function someOtherFunction()
    Wave wv = GetWave(param)

    // code
End
```

Reasons:

- More reliable to use
- Less code at the call site
- Avoids code duplication at the call site
- Allows to handle changes to the wave structure in one place (centralized resource management)

2.7 Unnecessary use of StringMatch

Bad:

```
if(StringMatch(str, ""))
    // code
endif

if(StringMatch(str, "abcd"))
    // code
endif
```

Good:

```
// can be written with a helper function if(isEmpty(str))
if(!cmpstr(str, ""))
    // ...
endif

// StringMatch is only required if you want to
// use ! or * in the second parameter
if(!cmpstr(str, "abcd"))
    // ...
endif
```

Reasons:

- Faster
- Better readability

2.8 Unnecessary loops

Bad:

```
for(i = 10; i < 101; i += 1)
    mywave[i] = i^2
endfor
```

Good:

```
mywave[10, 100] = p^2
```

Reasons:

- Much faster (at least ten times)

See also `DisplayHelpTopic "Waveform Arithmetic and Assignment"` for an in-depth introduction to the topic.

2.9 Unnecessary use of sprintf

Bad:

```
string str
sprintf str, "%s" GetName()
```

Good:

```
string str = GetName()
```

Reasons:

- Better readability
- Shorter
- Faster

2.10 Unnecessary use of Execute

Bad:

```
string cmd
sprintf cmd, "wv = 1 + 2"
Execute cmd
```

Good:

```
wv = 1 + 2
```

Reasons:

- Better readability
- Shorter
- Faster

2.11 Avoid relying on the top window and prefer structure based GUI control procedures

Bad:

```
Function ButtonProc(ctrlName) : ButtonControl
    String ctrlName

    GetWindow kwTopWin wtitle
    DoStuff(s_value)
End
```

Good:

```
// In case execution of this ButtonControl takes a long time
// and you want to prevent another call while the first is still
// progressing have a look at WMButtonAction::blockreentry
Function ButtonProc(ba) : ButtonControl
    struct WMButtonAction &ba

    switch(ba.eventCode)
        case 2:
            DoStuff(ba.win)
            break
    endswitch

    return 0
End
```

Reasons:

- Less error prone (the top window could be different if ButtonProc is called programmatically)
- Easily expandable to other events
- Faster as only a reference to the structure must be passed to the function

2.12 Avoid magic numbers

Bad:

```
settings[11][] = height
```

Good:

```
// either with file level constants
static Constant HEIGHT_ROW = 11

Function DoStuff()

    settings[HEIGHT_ROW][] = height

End

// or dimension labels, use SetDimLabel on the wave before
settings[%height][] = height

// or a set function (rarely an appropriate choice)
Function SetHeight(settings, height)
    WAVE settings
    variable height

    settings[11][] = height
End

SetHeight(settings, height)
```

Reasons:

- Better readability
- Less error prone

2.13 Unused parameters

Bad:

```
Function/S GetPath(param)
    string param

    return "root:myFolder"
End
```

Good:

```
Function/S GetPath()
    return "root:myFolder"
End
```

Reasons:

- Better readability
- Does not fake a dependency of the function upon the parameter

Note: Unused function variables and file level constants should also be avoided. As there is currently no support from Igor Pro in doing that, visual inspection must be performed.

2.14 Avoid function calls in loop statements

Bad:

```
for(i = 0; i < ItemsInList(list); i += 1)
    // code
endfor
```

Good:

```
numItems = ItemsInList(list)
for(i = 0; i < numItems ; i += 1)
    // code
endfor
```

Reasons:

- Faster, the bad example calls `ItemsInList` every time the loop condition is executed

3 Performance tests

3.1 Dimension labels versus numerical indizes

Using the following code

```
#pragma rtGlobals=3
#pragma igorVersion=6.3
#include <FunctionProfiling>

static Constant SIZE = 1e5

Function Prepare()
```

```
    variable i

    Make/O/N=(SIZE) data = p^2

    for(i = 0; i < SIZE; i += 1)
        SetDimLabel 0, i, $num2str(i), data
    endfor
End

Function Dimlabel()

    string str
    variable i, acc
    Wave data

    for(i = 0; i < SIZE; i += 1)
        str = num2str(i)
        acc += data[%$str]
    endfor

    print/D acc
    print str
End

Function NumericalIndex()

    string str
    variable i, acc
    Wave data

    for(i = 0; i < SIZE; i += 1)
        // The num2str call is included here in order to minimize
        // the number of differences compared to Dimlabel()
        str = num2str(i)
        acc += data[i]
    endfor

    print/D acc
    print str
End

Function PerformTest()
```

```
    Prepare()
    print "Array indexing with dimlabels"
    RunFuncWithProfiling(Dimlabel)

    print "Array indexing with numerical indizes"
    RunFuncWithProfiling(NumericalIndex)
End
```

we can compare the speed of dimension labels and numerical indizes. This approach asssumes that the `$` operator can be ignored in terms of execution speed.

Calling `PerformTest()`

```
Array indexing with dimlabels
333328333526912
99999
Total time=   21.39
Array indexing with numerical indizes
333328333526912
99999
Total time=   0.110777
```

clearly shows that numerical indizes are much faster than dimension labels.
Therefore you should use numeric indizes in performance critical code which usually is the case inside loops. In case you want to index a fixed element in a loop by dimension labels call `FindDimLabel` before the loop and store the numerical index.