OpenConsole Tools

These are a collection of tools and scripts to make your life building the OpenConsole project easier. Many of them are designed to be functional clones of tools that we used to use when developing inside the Windows build system.

Razzle

This is a script that quickly sets up your environment variables so that these tools can run easily. It's named after another script used by Windows developers to similar effect.

- It adds msbuild to your path.
- It adds the tools directory to your path as well, so all these scripts are easily available.
- It executes `\tools\.razzlerc.cmd` to add any other personal configuration to your environment as well, or creates one if it doesn't exist.
- It sets up the default build configuration to be 'Debug'. If you'd like to manually specify a build configuration, pass the parameter `dbg` for Debug, and `rel` for Release.

bcz

`bcz` can quick be used to clean and build the project. By default, it builds the `%DEFAULT_CONFIGURATION%` configuration, which is `Debug` if you use `razzle.cmd`.

- `bcz dbg` can be used to manually build the Debug configuration.
- `bcz rel` can be used to manually build the Release configuration.

opencon (and openbash, openps)

`opencon` can be used to launch the **last built** OpenConsole binary. If given an argument, it will try and run that program in the launched window. Otherwise it will default to cmd.exe.

`openbash` is similar, it immediately launches bash.exe (the Windows Subsystem for Linux entrypoint) in your `~` directory.

Likewise, `openps` launches powershell.

runformat & runxamlformat

`runxamlformat` will format `.xaml` files to match our coding style. `runformat` will format the c++ code (and will also call `runxamlformat`). **`runformat` should be called before making a new PR**, to ensure that code is formatted correctly. If it isn't, the CI will prevent your PR from merging.

The C++ code is formatted with `clang-format`. Many editors have built-in support for automatically running clang-format on save.

Our XAML code is formatted with XamlStyler. I don't have a good way of running this on save, but you can add a `git` hook to format before committing `.xaml` files. To do so, add the following to your `.git/hooks/pre-commit` file:

```
# XAML Styler - xstyler.exe pre-commit Git Hook
# Documentation: https://github.com/Xavalon/XamlStyler/wiki
# Originally from https://github.com/Xavalon/XamlStyler/wiki/Git-Hook
```

```
# Define path to xstyler.exe
XSTYLER_PATH="dotnet tool run xstyler --"

# Define path to XAML Styler configuration
XSTYLER_CONFIG="XamlStyler.json"

echo "Running XAML Styler on committed XAML files"
git diff --cached --name-only --diff-filter=ACM  | grep -e '\.xaml$' | \
# Wrap in brackets to preserve variable through loop
{
    files=""
    # Build list of files to pass to xstyler.exe
    while read FILE; do
        if [ "$files" == "" ]; then
            files="$FILE";
        else
            files="$files,$FILE";
        fi
    done

    if [ "$files" != "" ]; then
        # Check if external configuration is specified
        [ -z "$XSTYLER_CONFIG" ] && configParam="" || configParam="-c
$XSTYLER_CONFIG"

        # Format XAML files
        $XSTYLER_PATH -f "$files" $configParam

        for i in $(echo $files | sed "s/,/ /g")
        do
            #strip BOM
            sed -i '1s/^\xEF\xBB\xBF//' $i
            unix2dos $i
            # stage updated file
            git add -u $i
        done
    else
        echo "No XAML files detected in commit"
    fi

    exit 0
}
```

testcon, runut, runft

`runut` will automatically run all of the unit tests through TAEF. `runft` will run the feature tests, and `testcon` runs all of them. They'll pass any arguments through to TAEF, so you can more finely control the testing.

A recommended workflow is the following command:

```
bcz dbg && runut /name:*<name of test>*
```

Where `<name of test>` is the name of the test testing the relevant feature area you're working on. For example, if I was working on the VT Mouse input support, I would use `MouseInputTest` as that string, to isolate the mouse input tests. If you'd like to run all the tests, just ignore the `/name` param: `bcz dbg && runut`

To make sure your code is ready for a pull request, run the build, then launch the built console, then run the tests in it. The built console will inherit all of the razzle environment, so you can immediately start using the macros:

1. `bcz`
2. `opencon`
3. `testcon` (in the new console window)
4. `runformat`

If they all come out green, then you're ready for a pull request!