Getting Started with Code Analyzer Configuration

The Code Analyzer checks code and provides information about errors and warnings. You can configure the Code Analyzer to check for additional issues, modify the severity or message text of existing checks, or even disable checks for specific issues.

This example will use the provided `codeAnalyzerConfiguration.json` file to:

1) Add additional checks:

- Disallow "`eval`" and "`evalc`" usage
- Disallow very long functions (2000+ lines)
- Disallow very long lines (1000+ characters)
- Disallow deeply nested control statements (10+ nested control statements)
- Warn when a function has a lot of input arguments (10+ input arguments)
- Warn when a function has a lot of output arguments (10+ output arguments)

2) Raise the severity of some existing checks from "`warning`" to "`error`:"

- Disallow the use of global variables
- Raise the severity of the built-in warning against the use of "`eval`" and "`evalc`"
- Disallow the use of "`ans`" as a variable
- Disallow chained logical expressions (e.g., `a > b > c`)

Let's get started!

Analyze code issues before applying the Code Analyzer configuration

First, we'll remove the Code Analyzer configuration if it exists.

```
if isfile("resources/codeAnalyzerConfiguration.json")
    delete("resources/codeAnalyzerConfiguration.json")
end
```

Now, let's analyze our code for issues.

```
codeIssues
```

```
ans =
```

```
  codeIssues with properties:

                      Date: 12-Jun-2023 19:26:20
                   Release: "R2023a"
                     Files: [10×1 string]
    CodeAnalyzerConfiguration: "active"
                    Issues: [4×10 table]
             SuppressedIssues: [0×11 table]

    Issues table preview

              Location                Severity    Fixability
    ____________________________    ________    __________    ____________________________________________________
    "ansExample.m"                  warning       manual      "Using ANS as a variable is not recommended as ANS is
    "chainedComparisonsExample.m"   warning       manual      "Expressions like a > b > c are interpreted as (a > b
    "evalExample.m"                 warning       manual      "'eval' is inefficient and makes code less clear. Cal
    "globalExample.m"               warning       manual      "Global variables are inefficient and make errors dif
```

It looks like our code already has some existing warnings. Some of these warnings are for potential bugs, like in the **`chainedComparisonsExample.m`** file, but there might be other issues we may want to have Code Analyzer identify for us.

Applying a more strict Code Analyzer configuration

We can apply Code Analyzer configuration by copying the **`codeAnalyzerConfiguration.json`** file to the resources directory.

```
if ~isfolder("resources")
    mkdir("resources");
end
copyfile("codeAnalyzerConfiguration.json","resources");
```

Now, let's analyze our code for issues using our new Code Analyzer configuration.

```
codeIssues
```

```
ans =
  codeIssues with properties:

                      Date: 12-Jun-2023 19:26:20
                   Release: "R2023a"
                     Files: [10×1 string]
    CodeAnalyzerConfiguration: "active"
                    Issues: [10×10 table]
             SuppressedIssues: [0×11 table]

    Issues table preview

                  Location                    Severity    Fixability
    ____________________________________    ________    __________    ____________________________________________
```

```
"ansExample.m"                              error         manual      "Using ANS as a variable is not recommended
"chainedComparisonsExample.m"               error         manual      "Expressions like a > b > c are interpreted
"deeplyNestedControlStatementExample.m"     error         manual      "This control statement is deeply nested (n
"evalExample.m"                             error         manual      "The use of "eval" and "evalc" is forbidden
"evalExample.m"                             error         manual      "'eval' is inefficient and makes code less
"functionTooLongExample.m"                  error         manual      "Function has more than 2000 lines. This ma
"globalExample.m"                           error         manual      "Global variables are inefficient and make
"lineTooLongExample.m"                      error         manual      "Line has more than 1000 characters (includ
"tooManyInputsExample.m"                    warning       manual      "Function has more than 10 input arguments.
"tooManyOutputsExample.m"                   warning       manual      "Function has more than 10 output arguments
```

Wow! That's a lot more issues!

But that's exactly what we want! We want to identify these kinds of coding patterns, and address them before they end up in our production code.

Now try using this Code Analyzer configuration in your codebase

Now that you know how the Code Analyzer configuration works, try applying this Code Analyzer configuration to your existing code to see if you find any new code issues. And feel free to customize this configuration file to fit your needs.

Enjoy writing better MATLAB code!