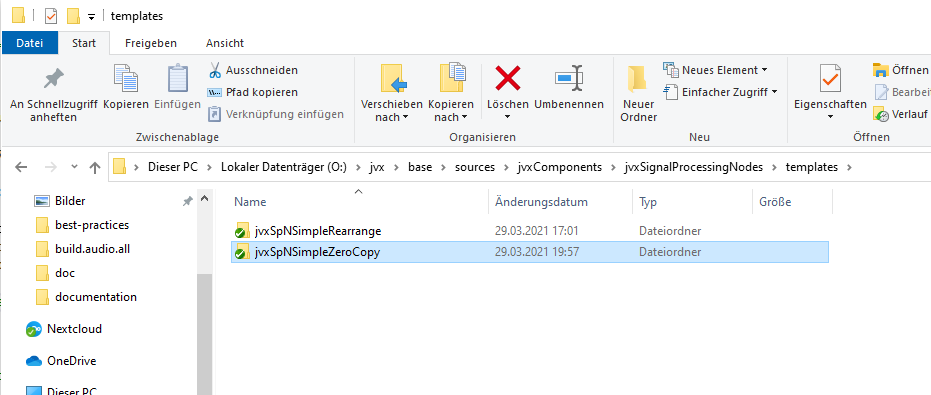
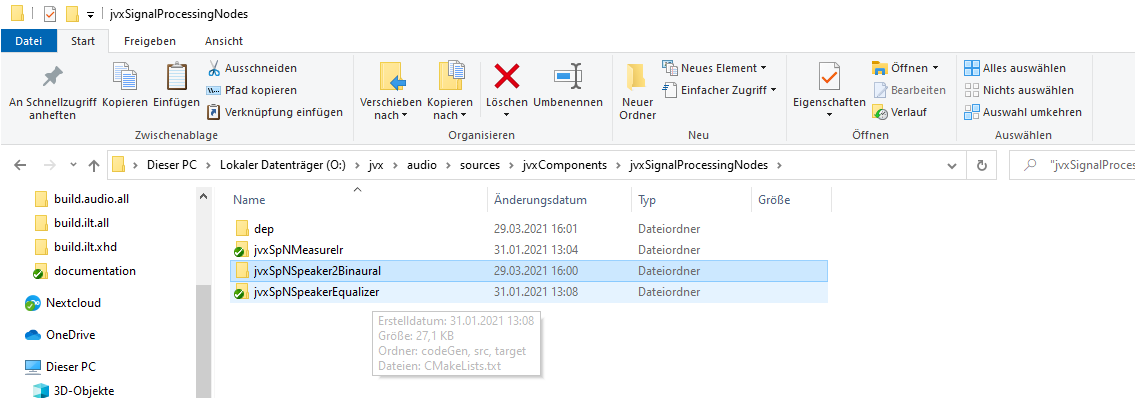
# How to create a new ZERO-COPY Signal Processing Node

## Step I: Copy working project

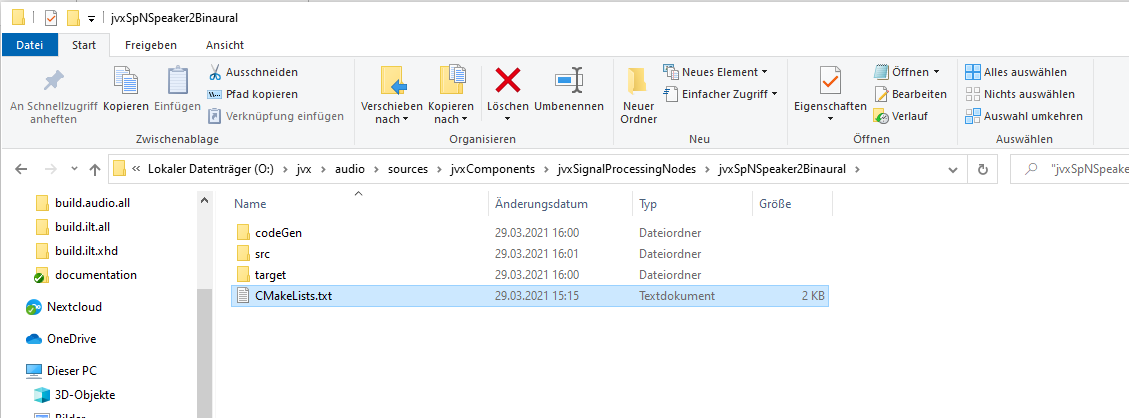


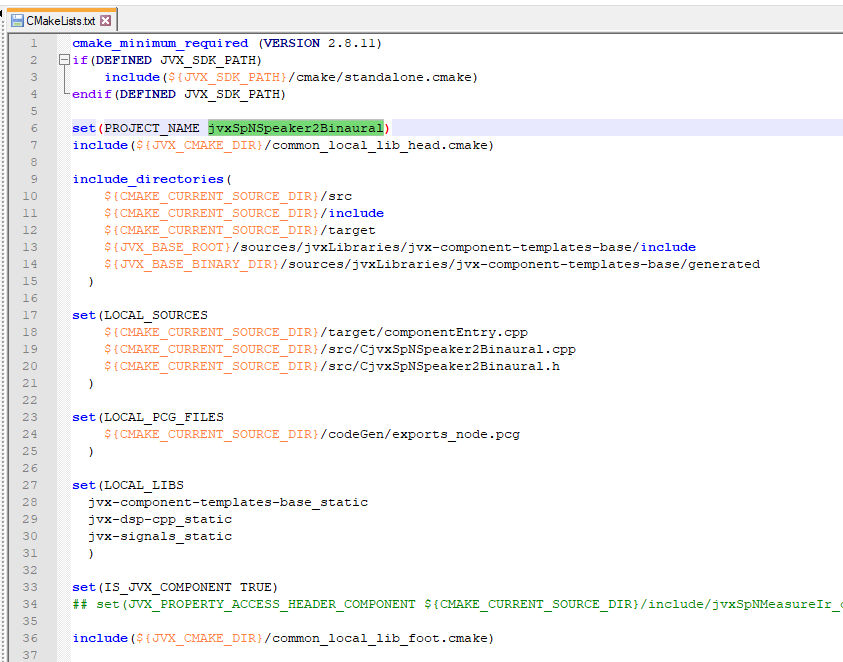
## Step II: Rename all files and folders





## Step III: Modify CMakeLists.txt to define a new project



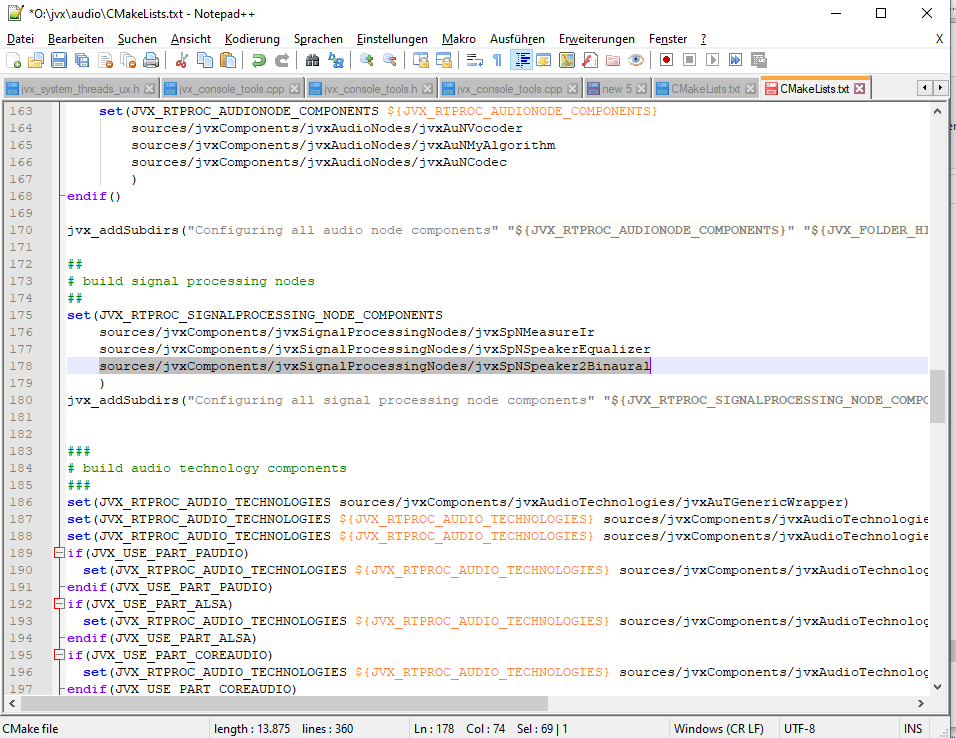


## Step IV: Reference the souce code files



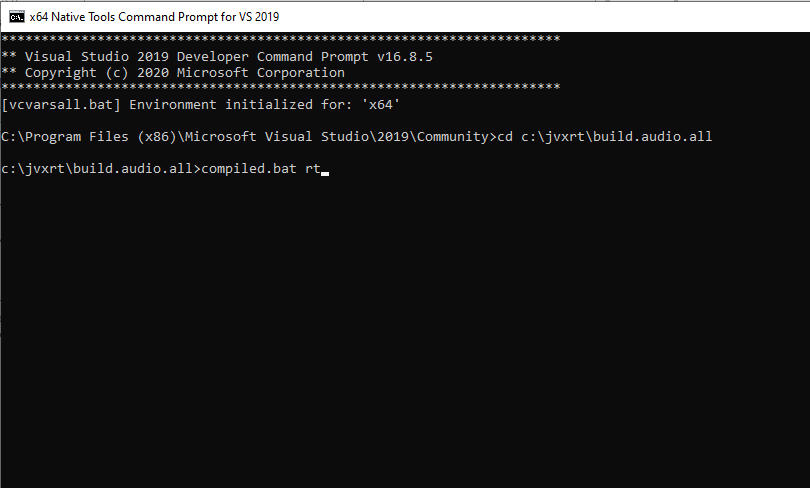
## Step V: Modify the project build files to integrate the new component

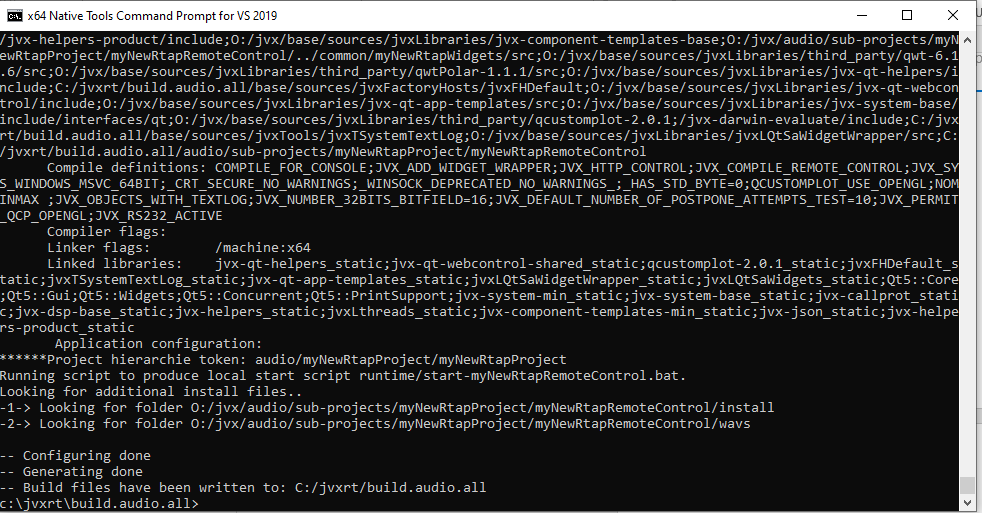
## 



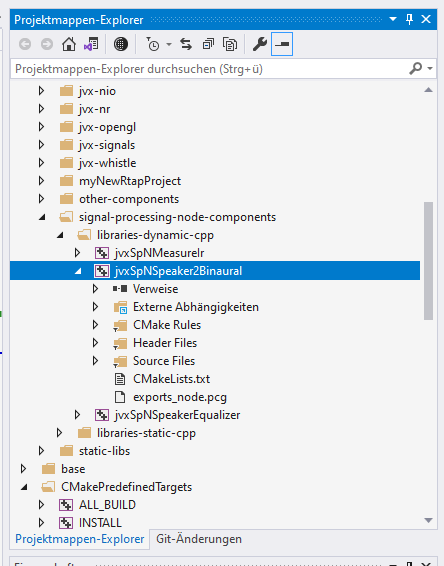
## Step VI: Run project configuration



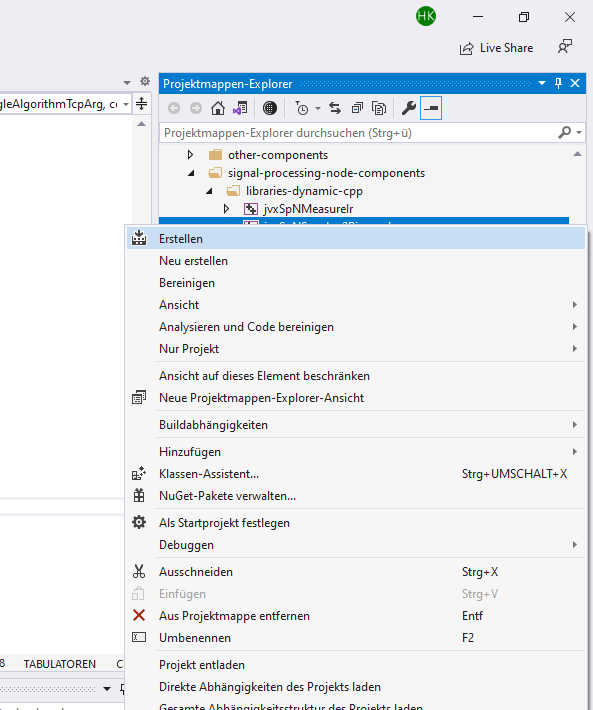




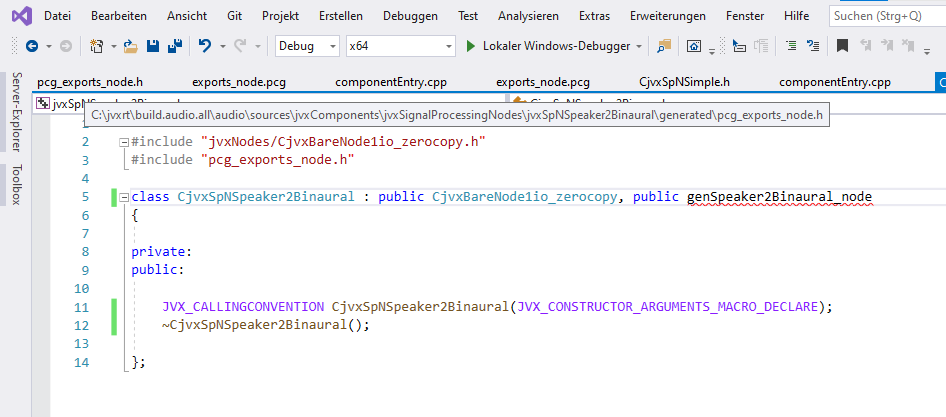
## Step VII: Re-open project in Visual Studio

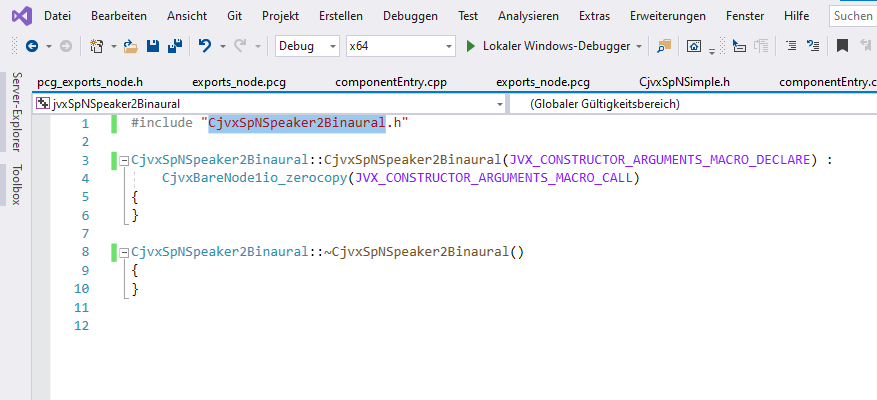


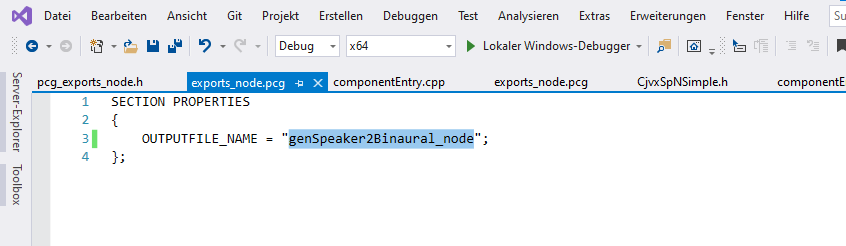
## Step VIII: First Build of the new project



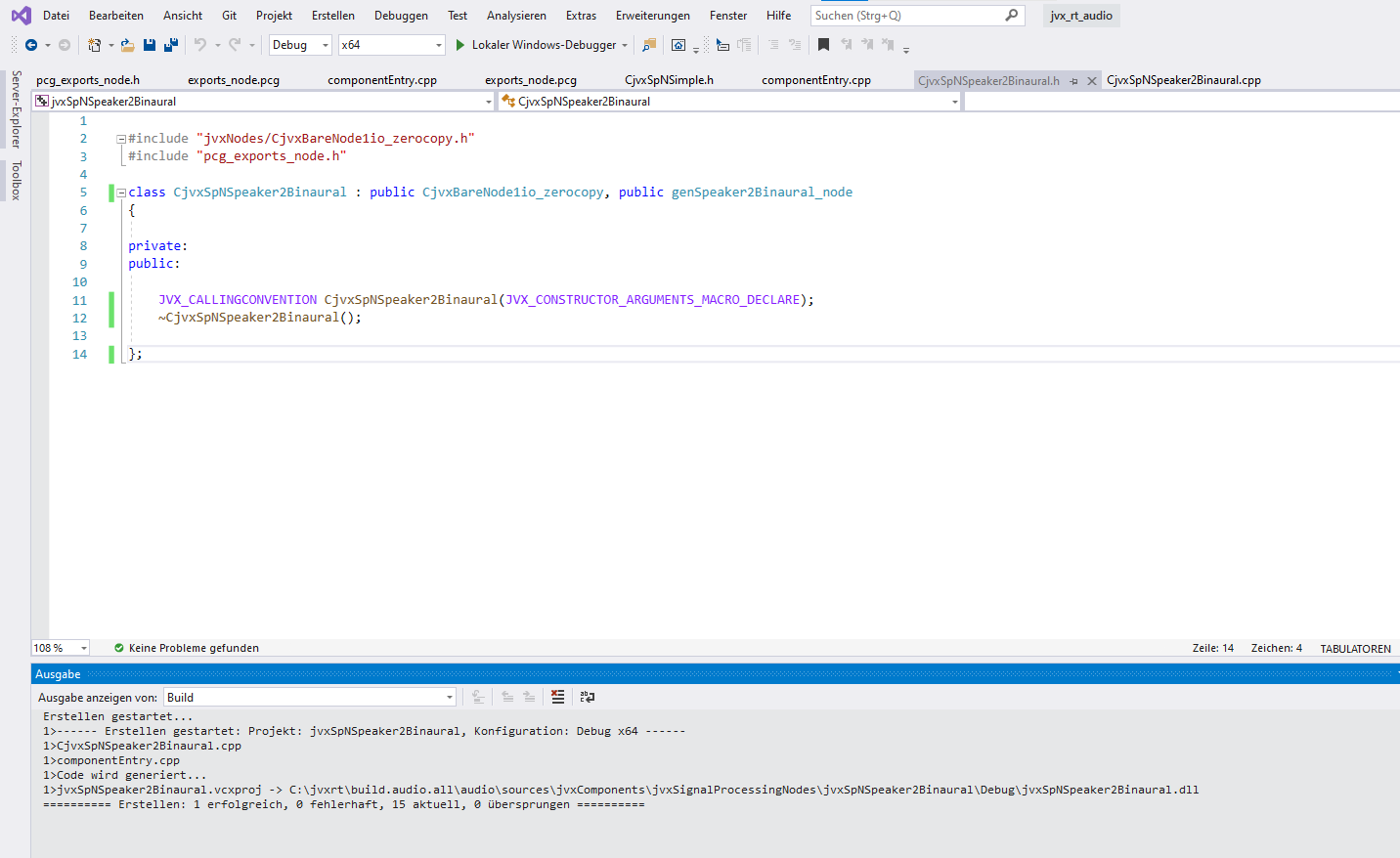
## Step IX: Adaptation of source code



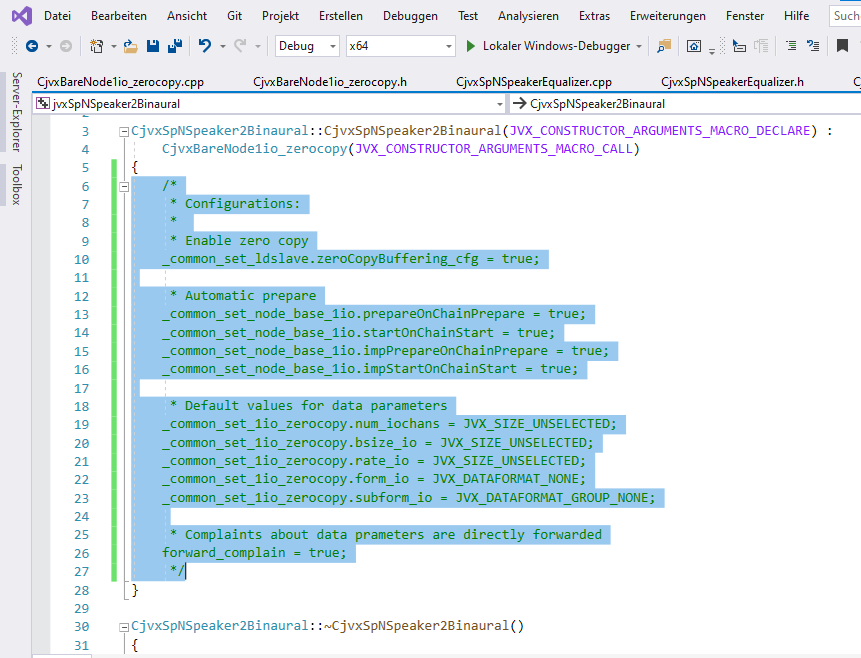




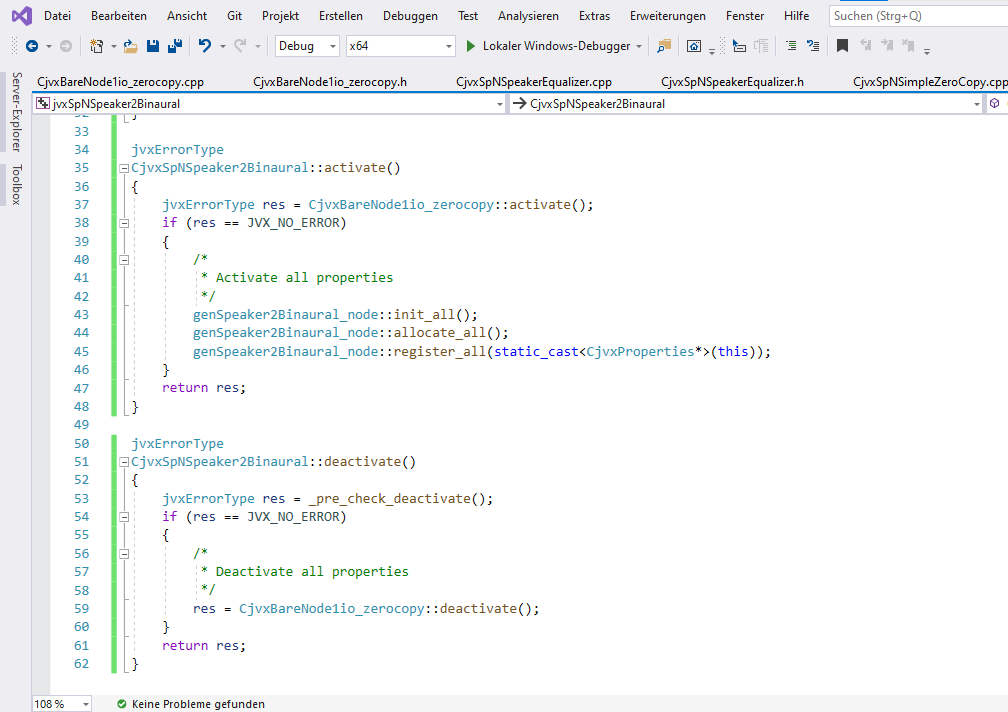
## Step X: Build the project with latest code adaptations



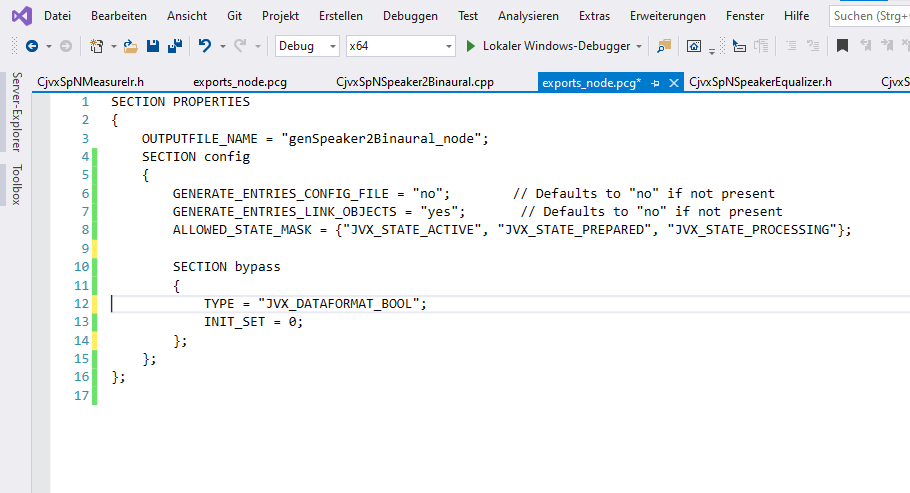
## Step XI: Configure behavior of base class



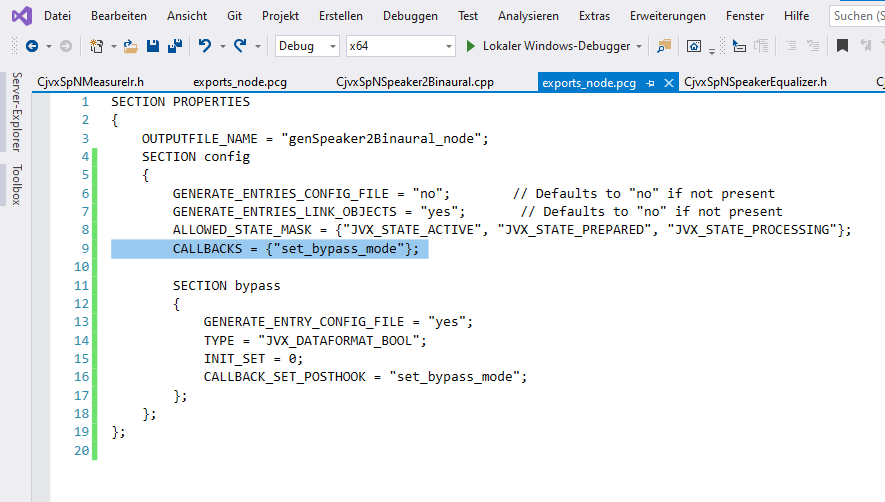
## Step XII: Activate Properties



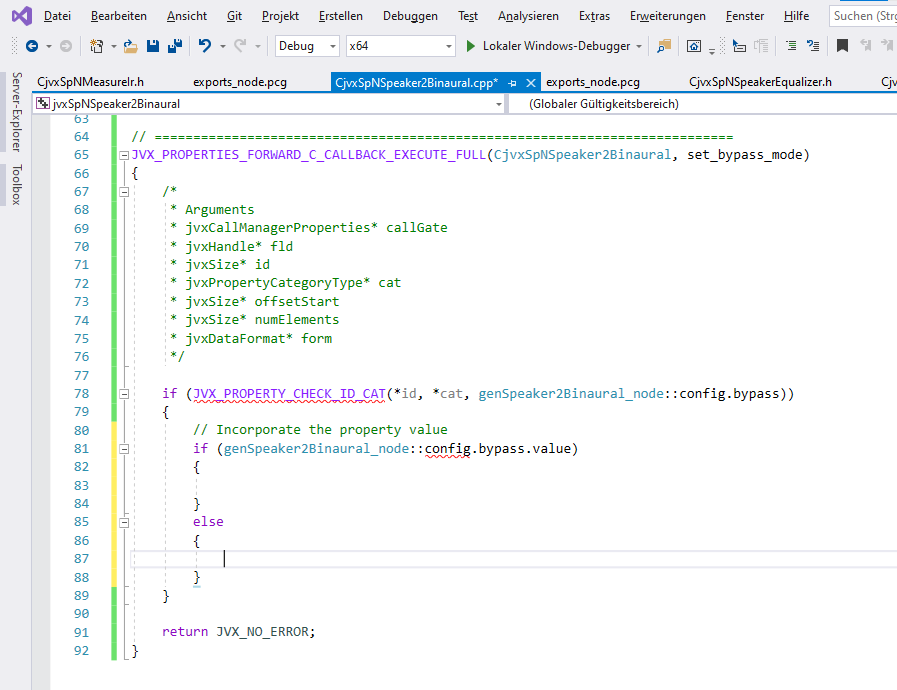
## Step XIII: Specify first property “bypass”



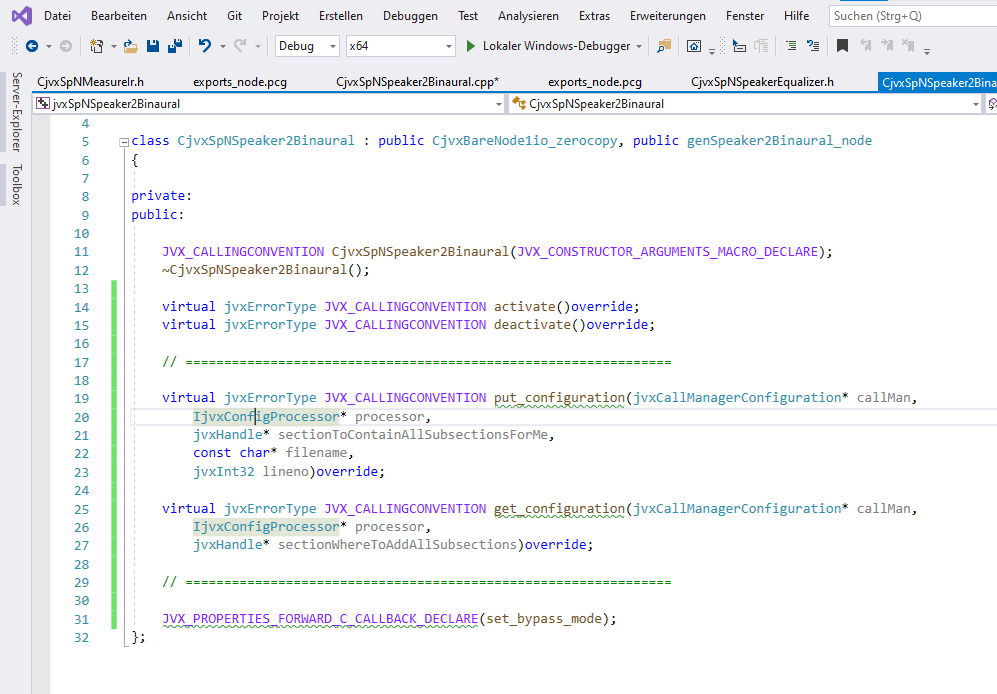
## Step XIV: Add storage of property in config file and callback to react to new property values

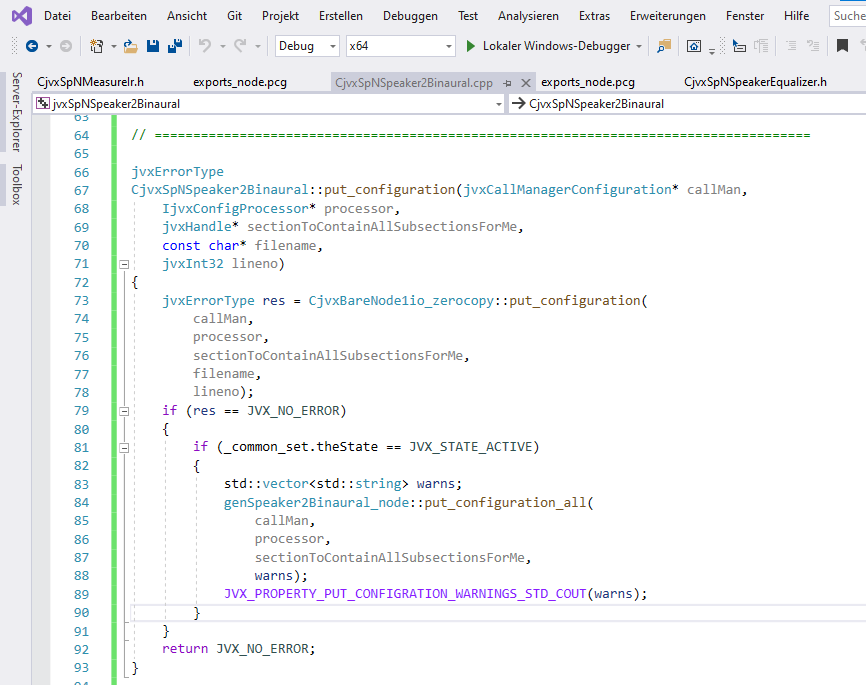


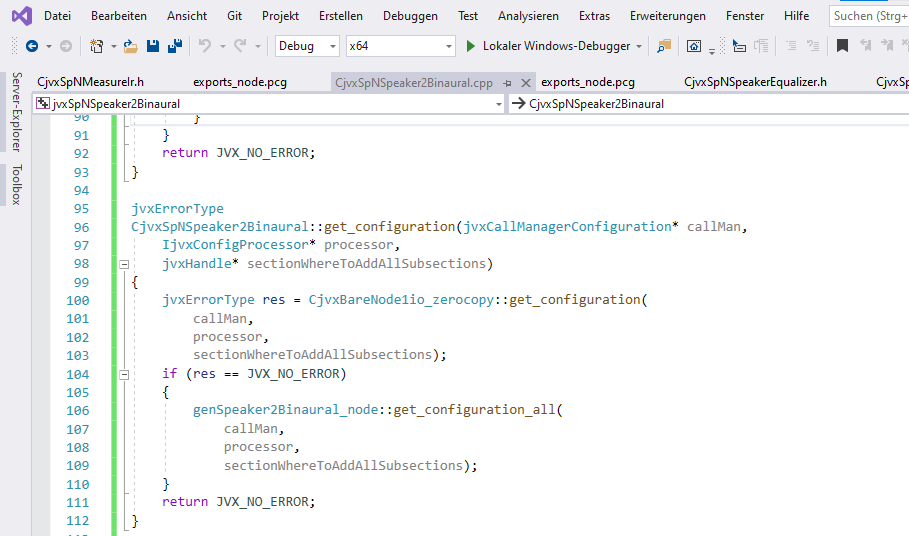
## Step XV: Implement callback for property on new value



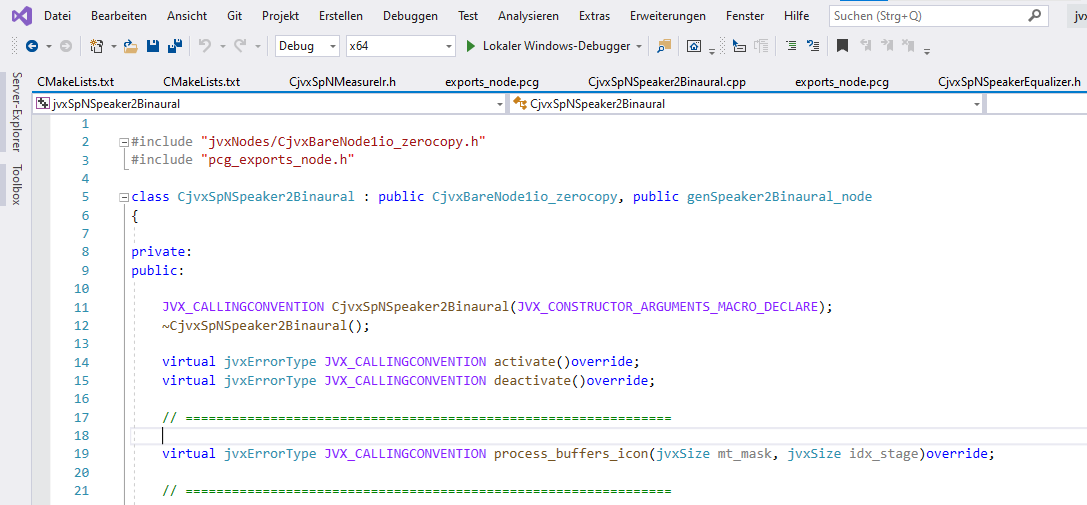
## Step XVI: Implement read/write of properties from /to configuration file

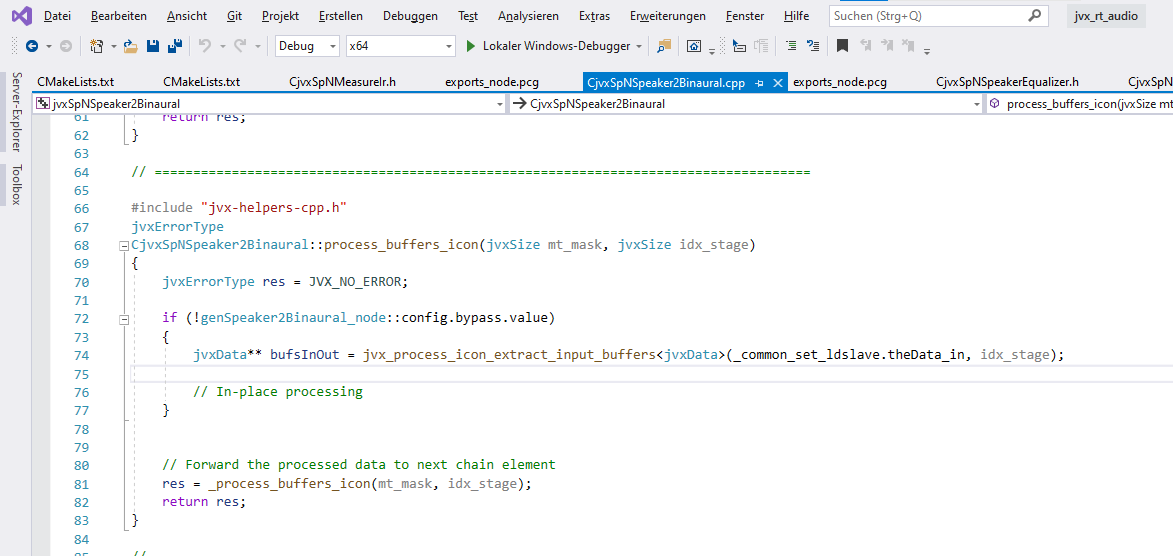






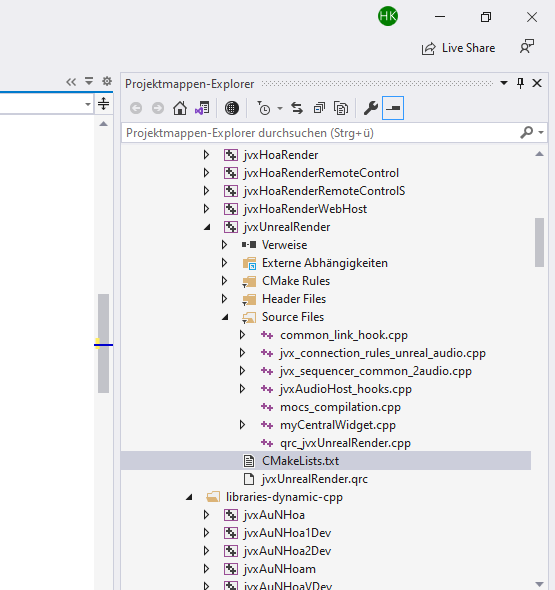
## Step XVII: Implement data processing callback

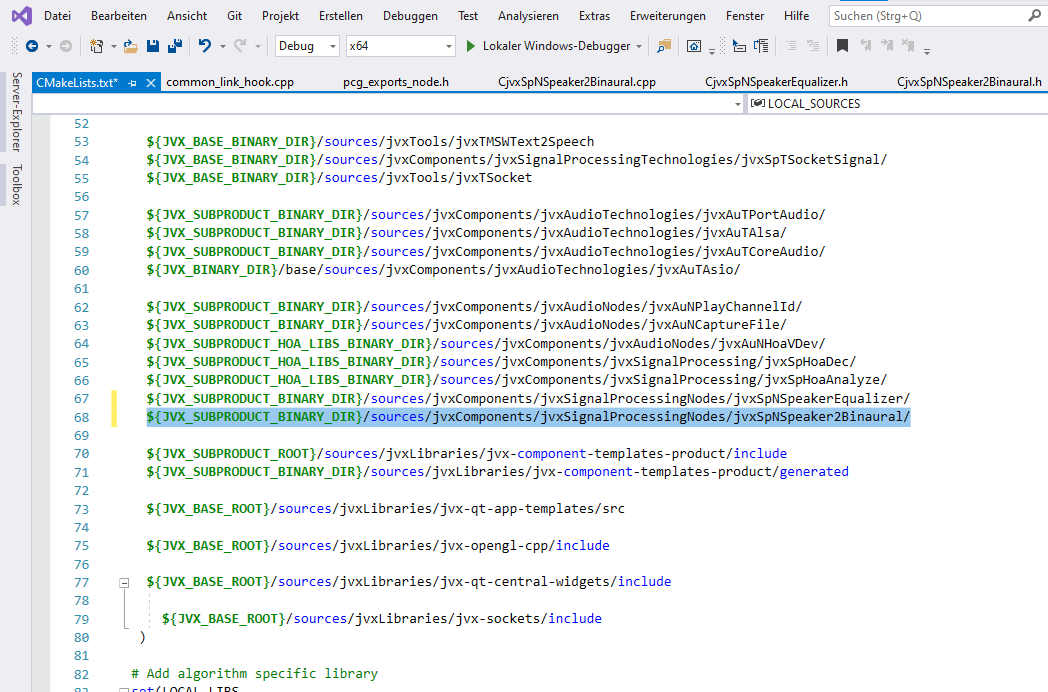


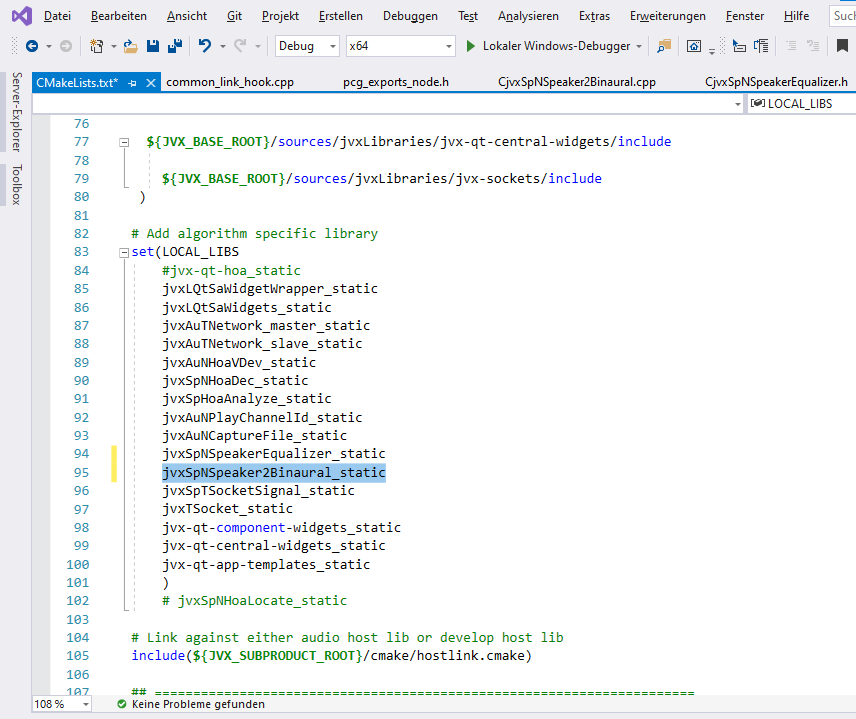


## Step XVIII: Add signal processing node to project executable, e.g. jvxUnrealRender

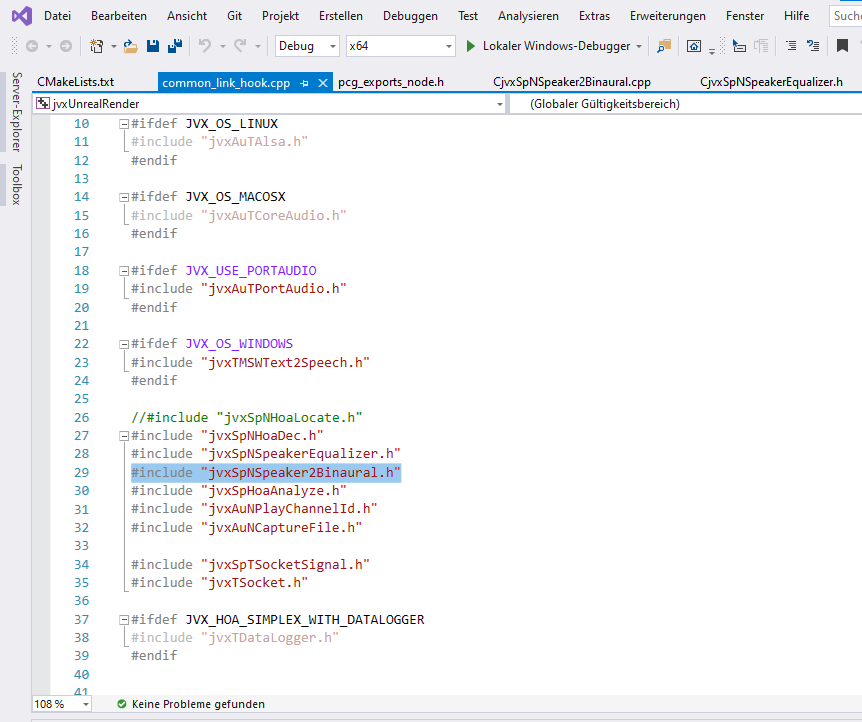
1. Add module to CMakeLists.txt: header include and library

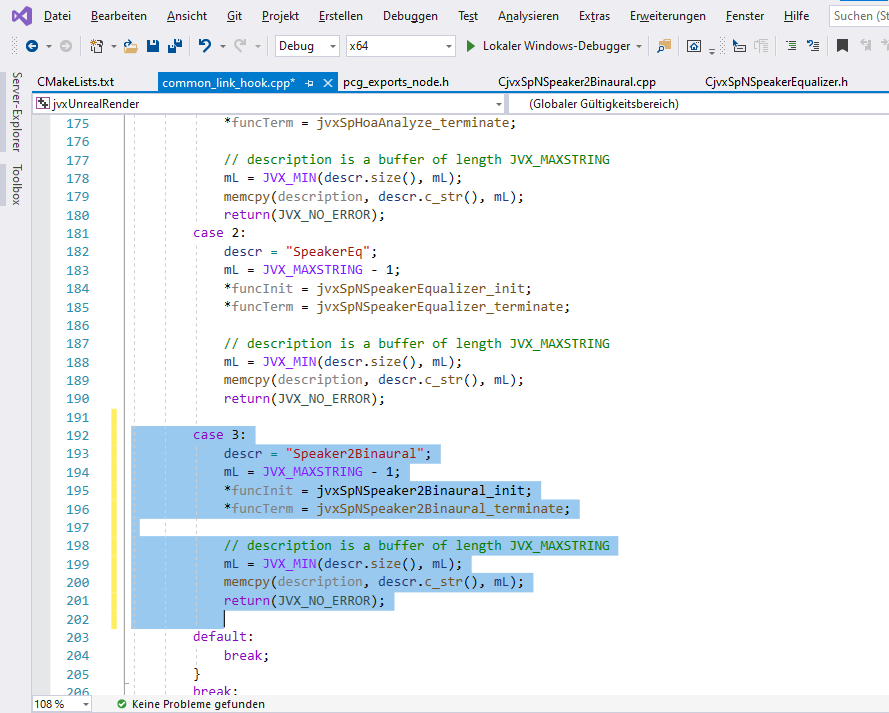




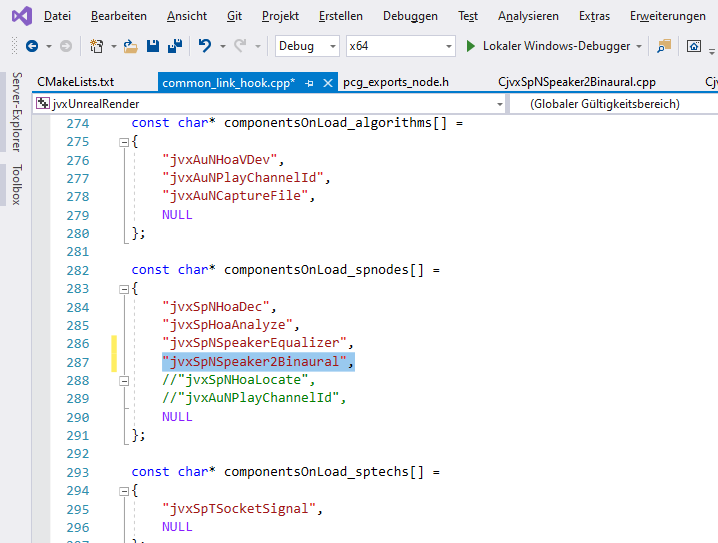


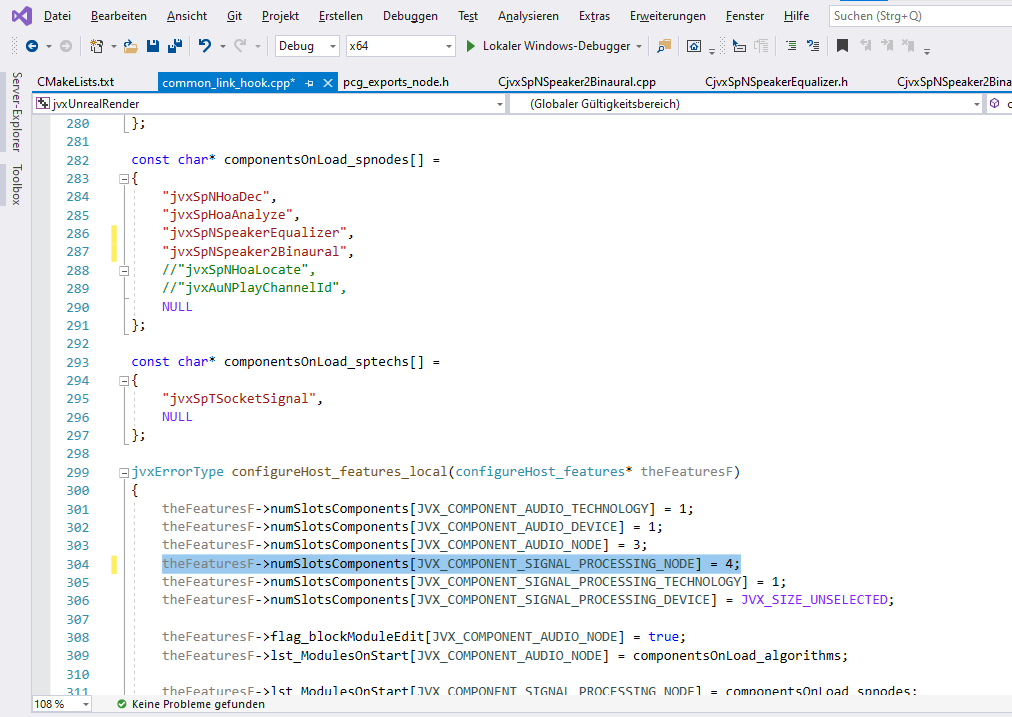
1. Add entry to hook to load component library at startup, include and code





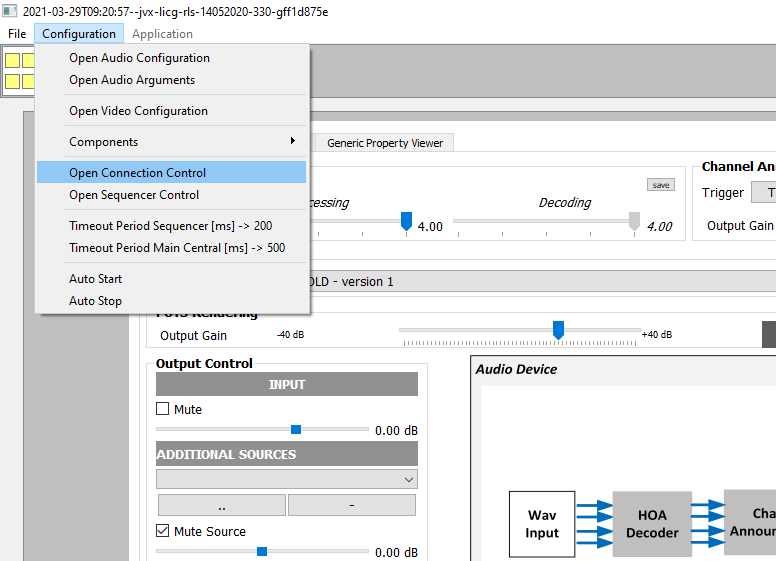
1. Add code to auto-start the new component, entry in table and number of slots





## Step XIX: Review the signal processing component

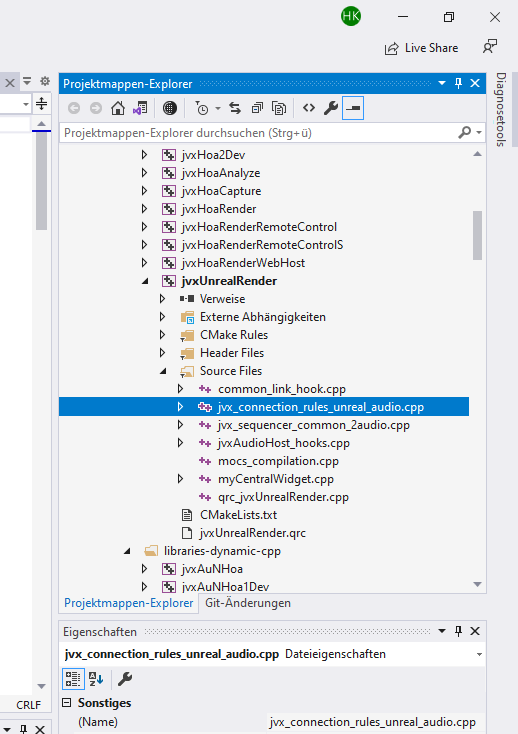


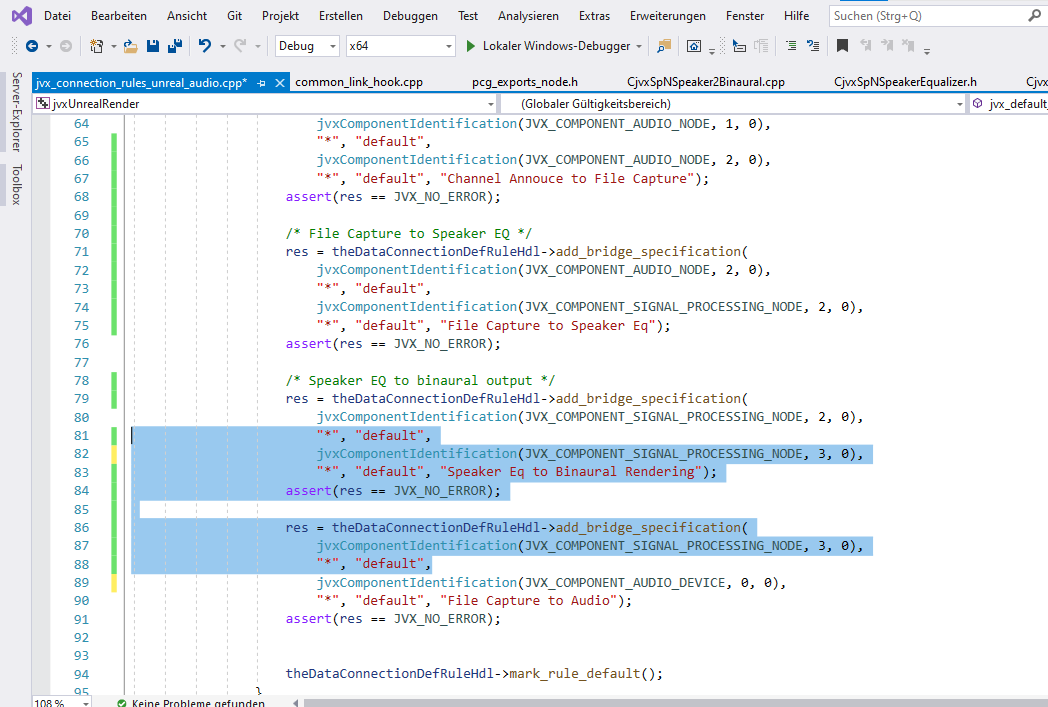


Component not yet involved in signal processing chain:

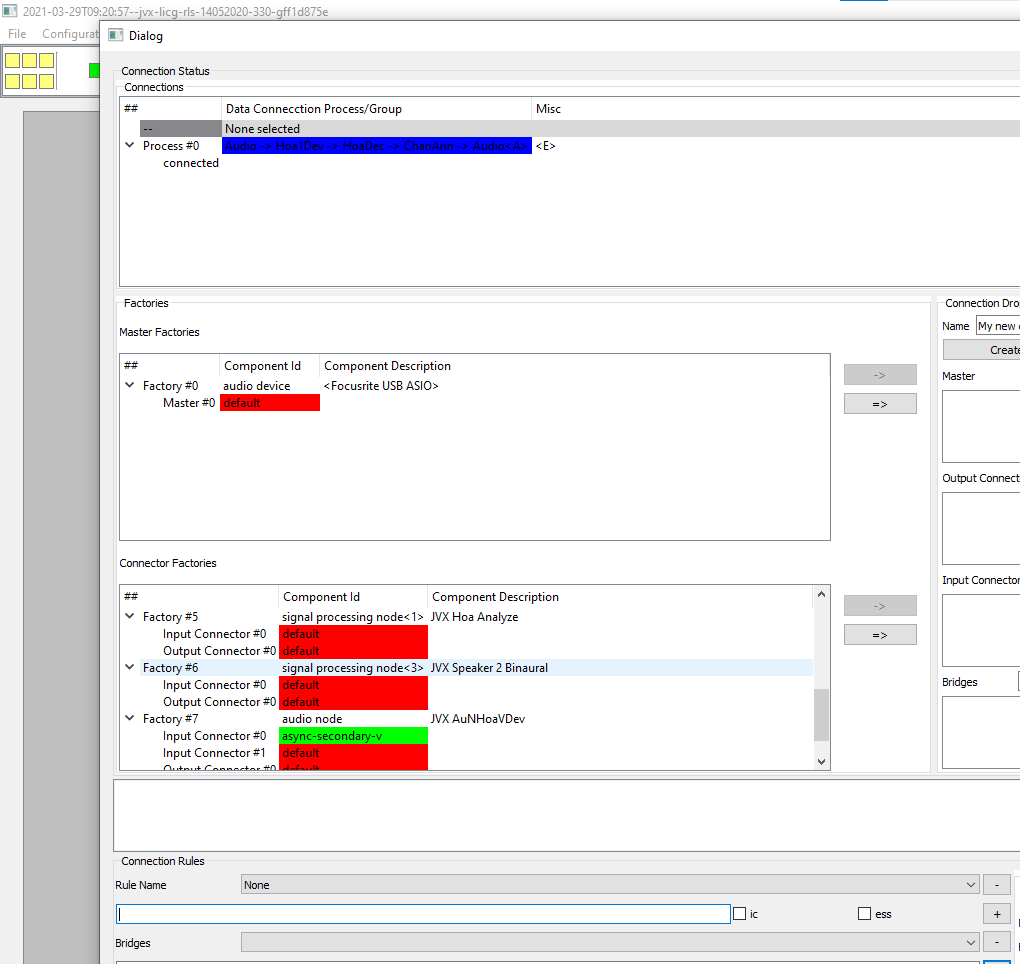


## Step XX: Extend connection to support binaural renderer

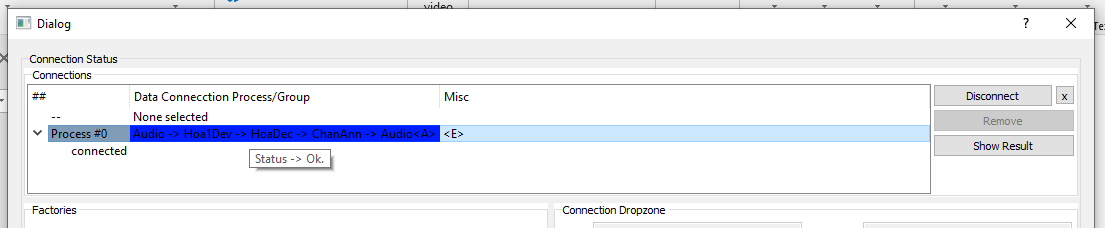




## Step XXI: Review of new connections



Review of the data flows

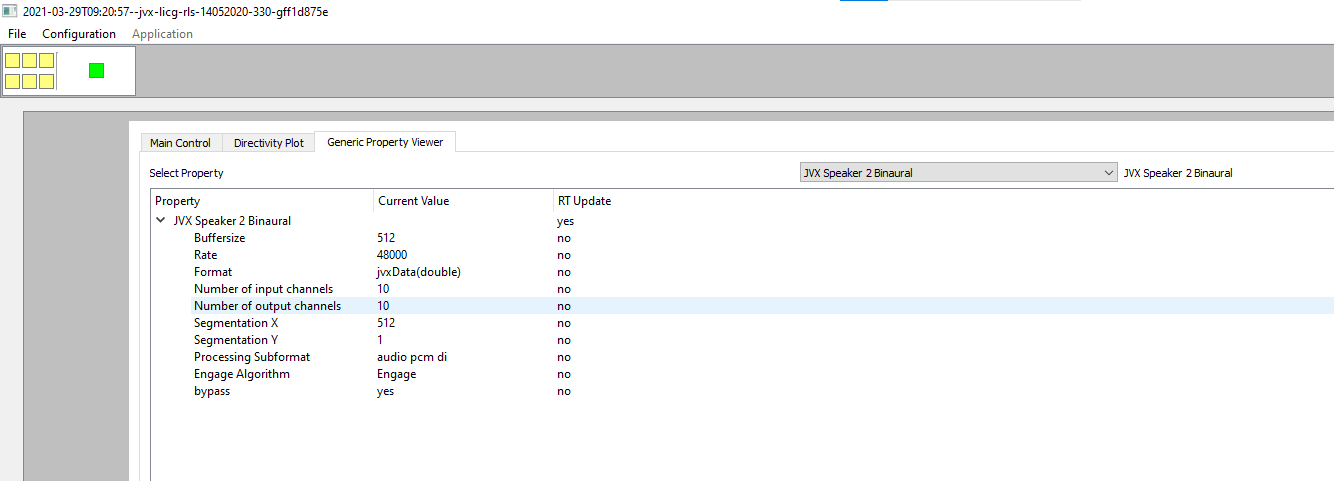




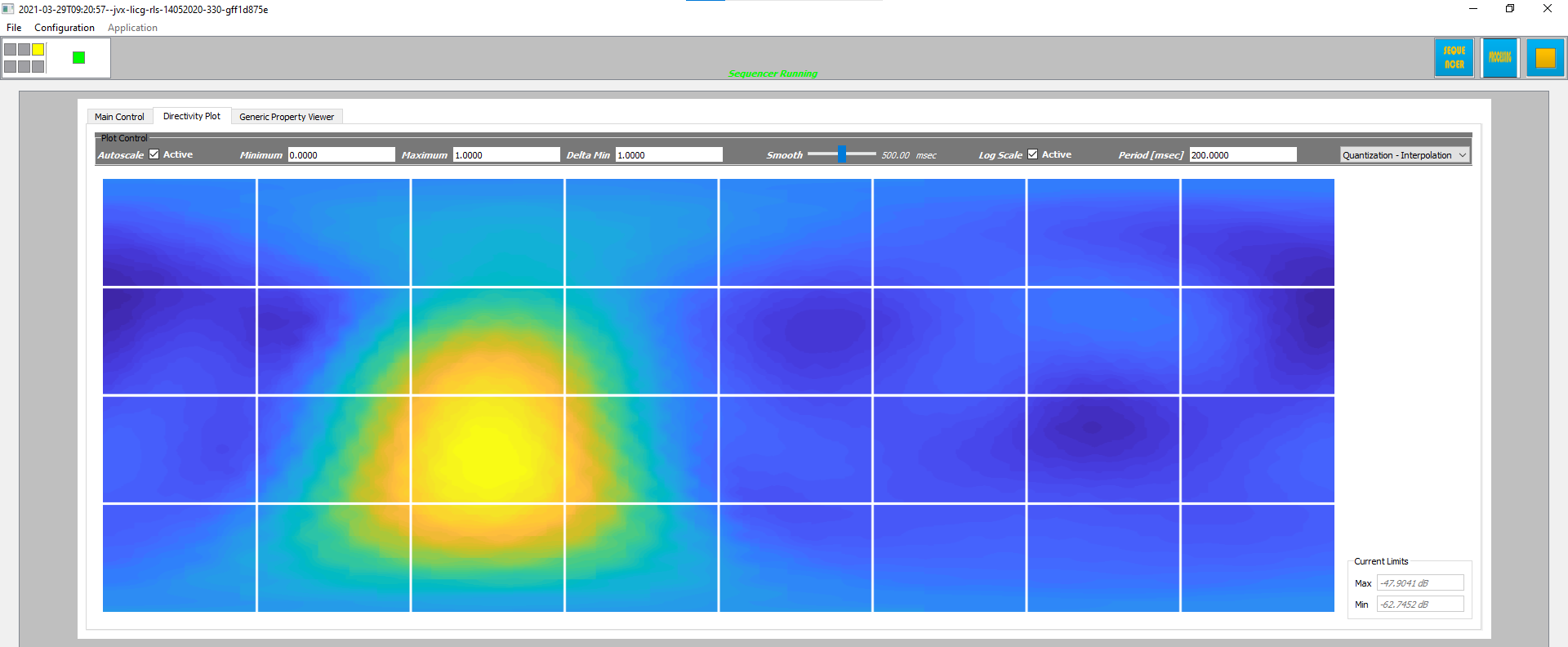
New component inserted right before output to audio device



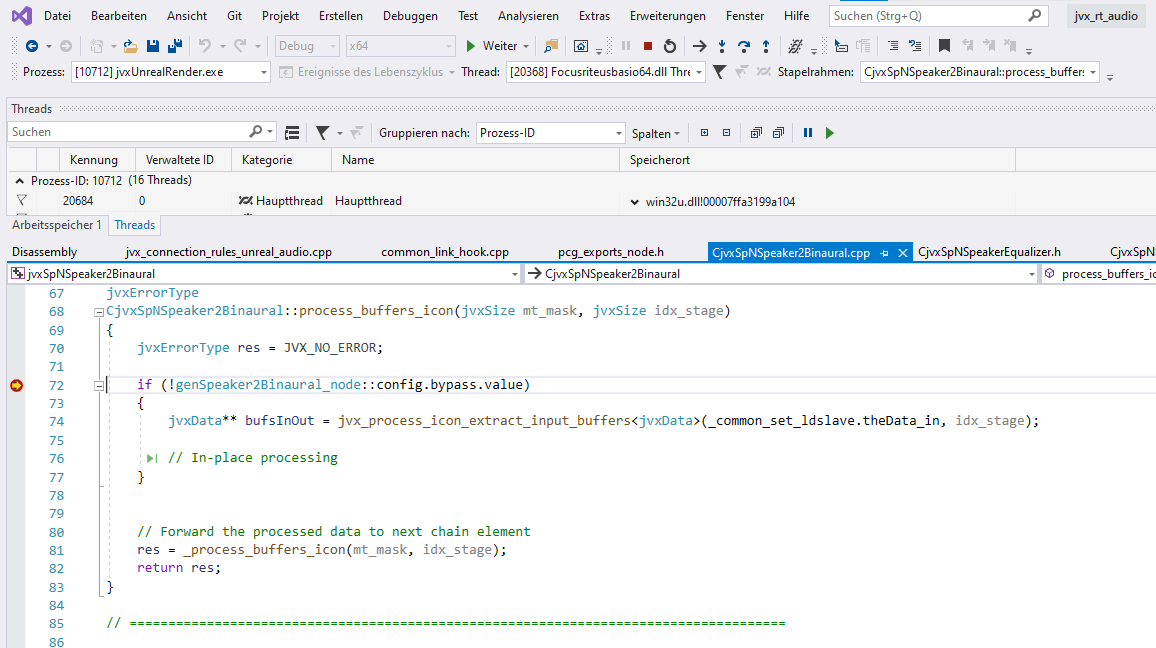
## Step XXII: Review Connection Parameters

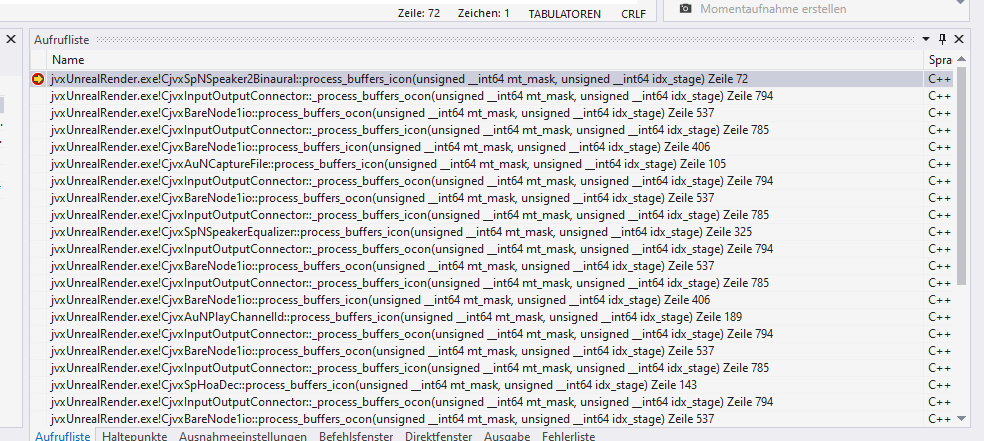


## Step XXIII: Run Processing with new component

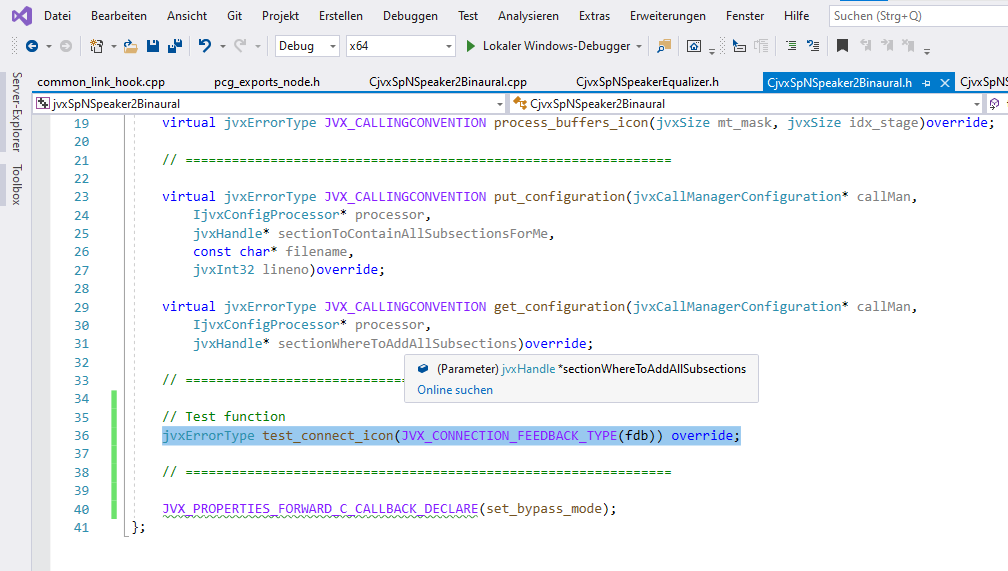


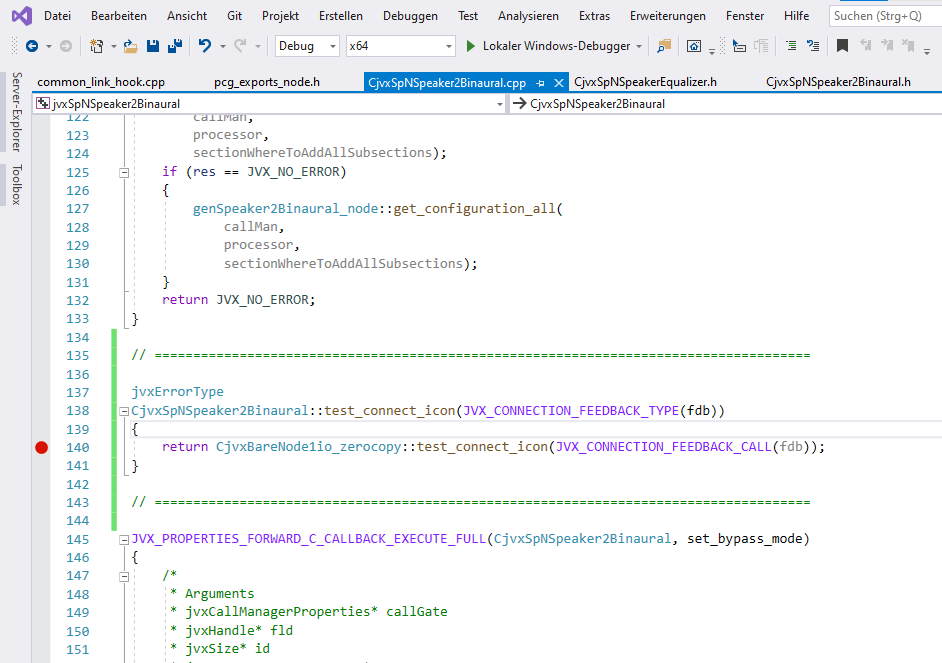
## Step XXIV: Debug Processing Function





## Step XXV: Setup connection constraints



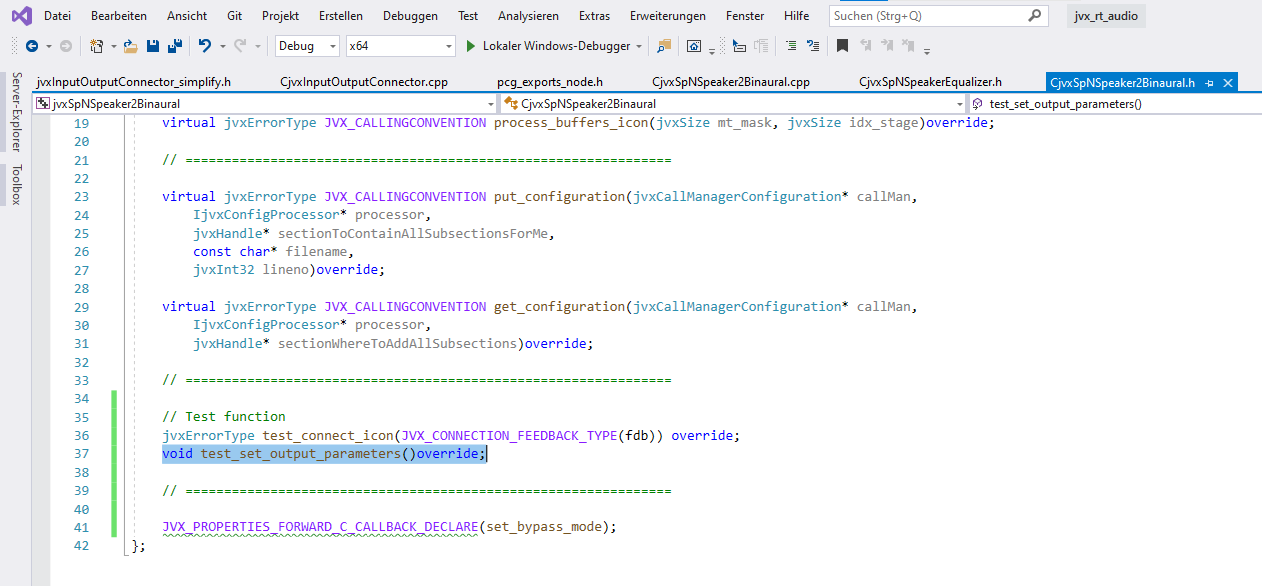


Default behavior: Input Parameters = Output Parameters (ZERO COPY MODULE!)

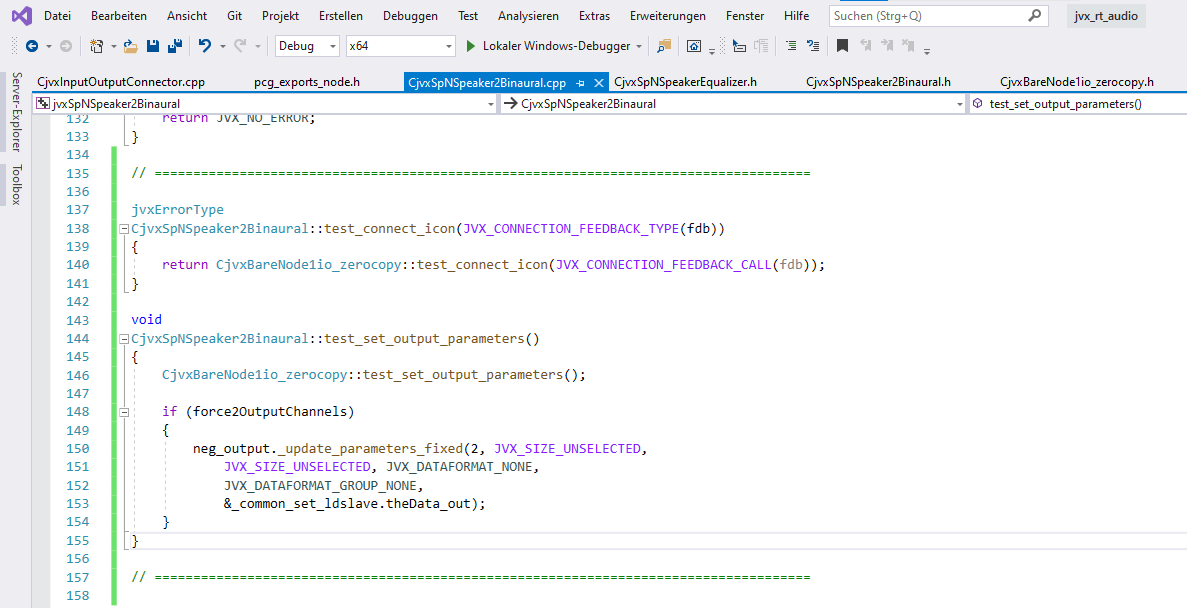
## Step XXVI: Deviate from default behavior: modify number of output channels

* Source Code to view:



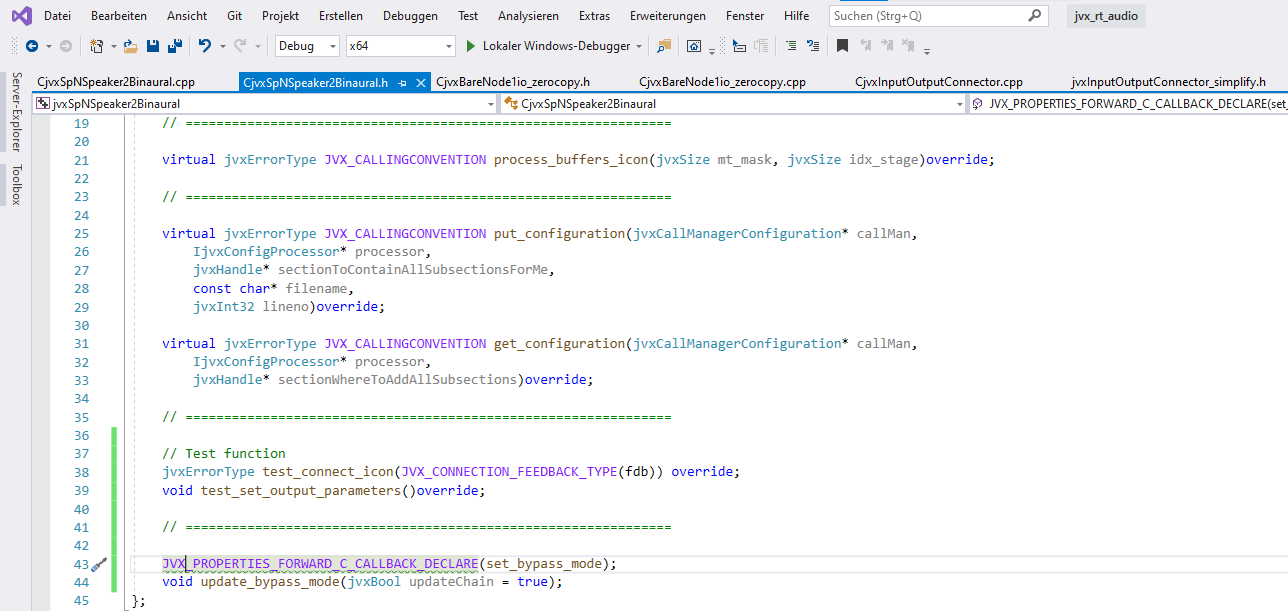


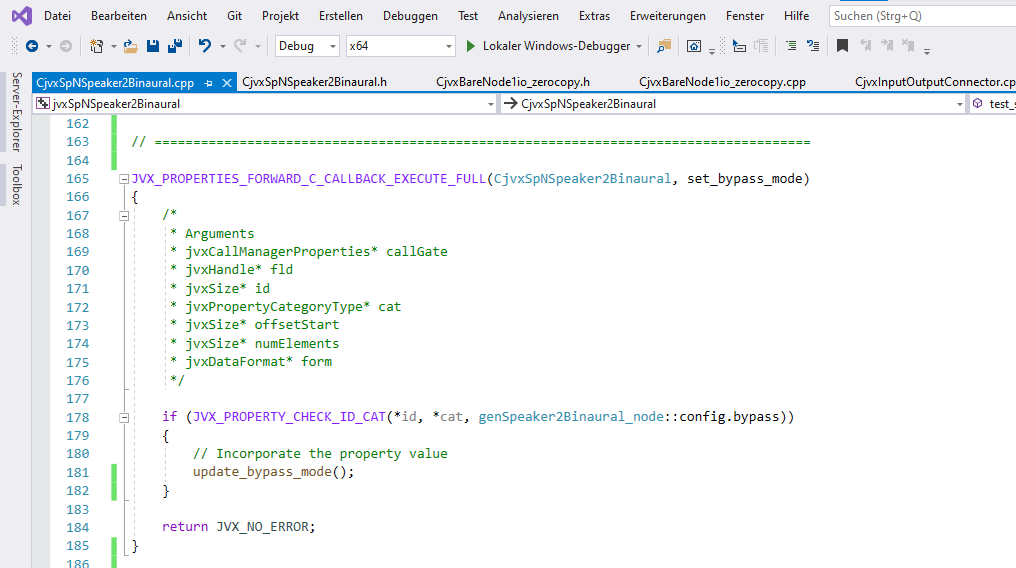
* Specify two output channels in specific mode – bool *force2OutputChannels*

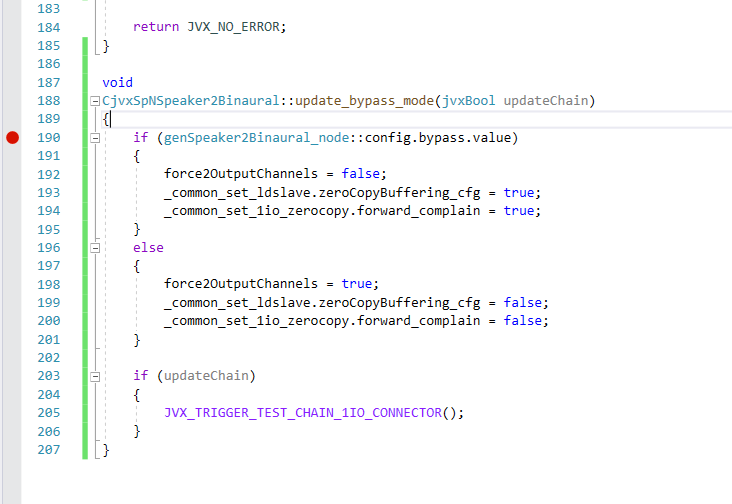


*neg\_output* holds acceptable setting ranges whereas *\_common\_set\_ldslave.theData\_out* specifies a single, concrete value.

## Step XXVI: Make it configurable in UI: connect property





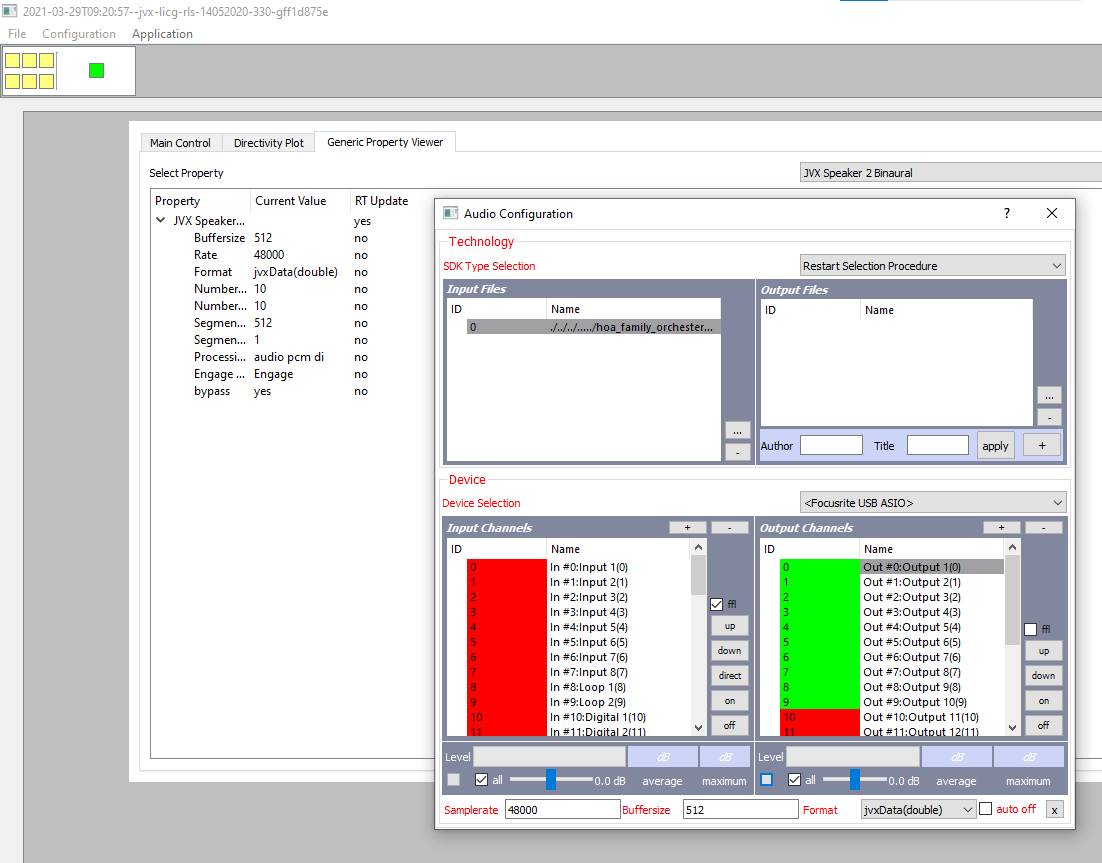
****

no zerocopy  
Format change (channel number) change in module

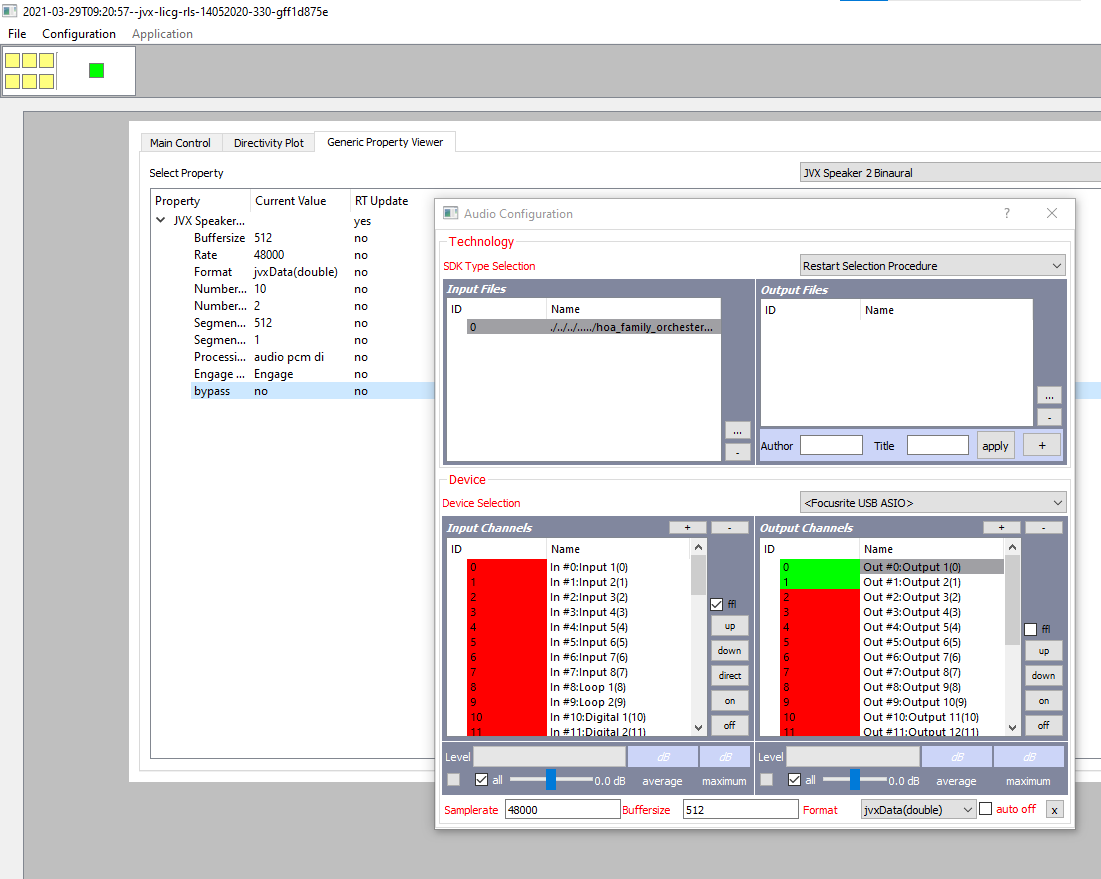
Bypass = zerocopy  
Parameter change with impact on successor

## Step XXVII: Test variable settings

Option 1: Bypass mode on:

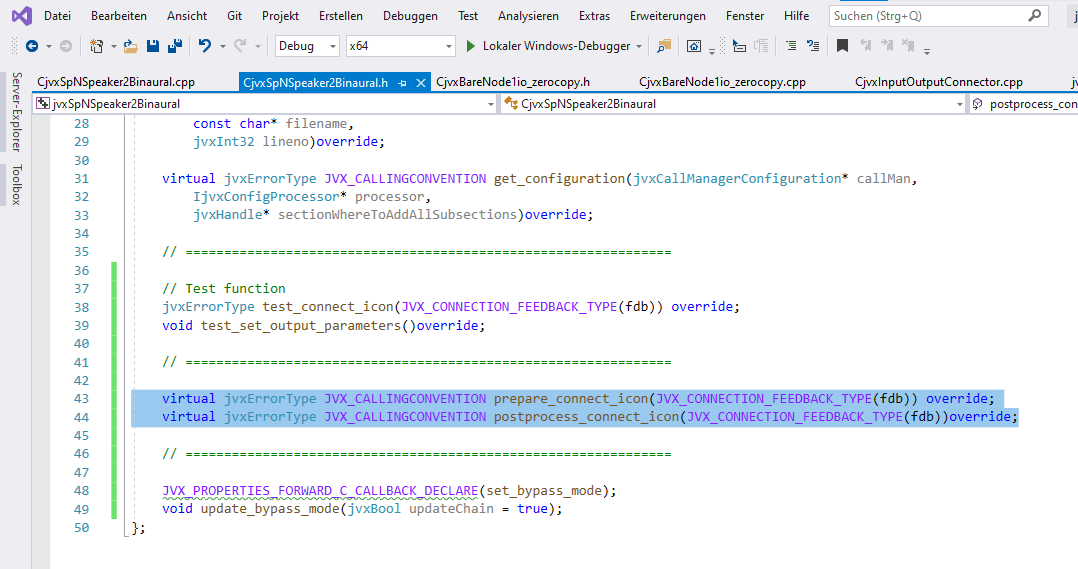


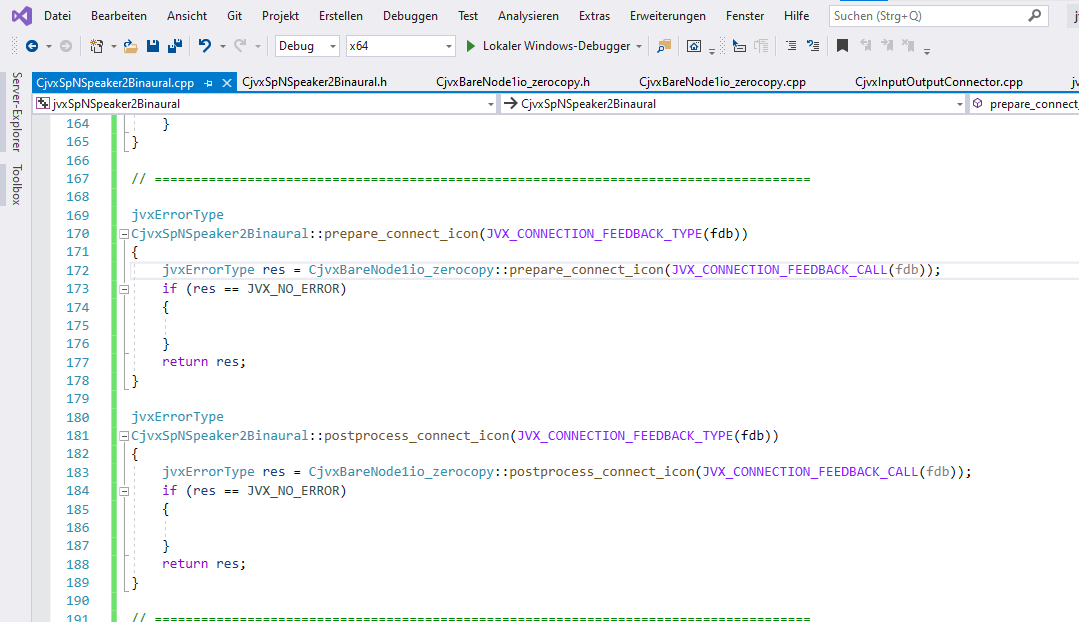
Option 1: Bypass mode off:



## Step XXVIII: Setup Data Processing

Indeed, this is not required since base class does everything already.





## Step XXIX: Implement signal processing

