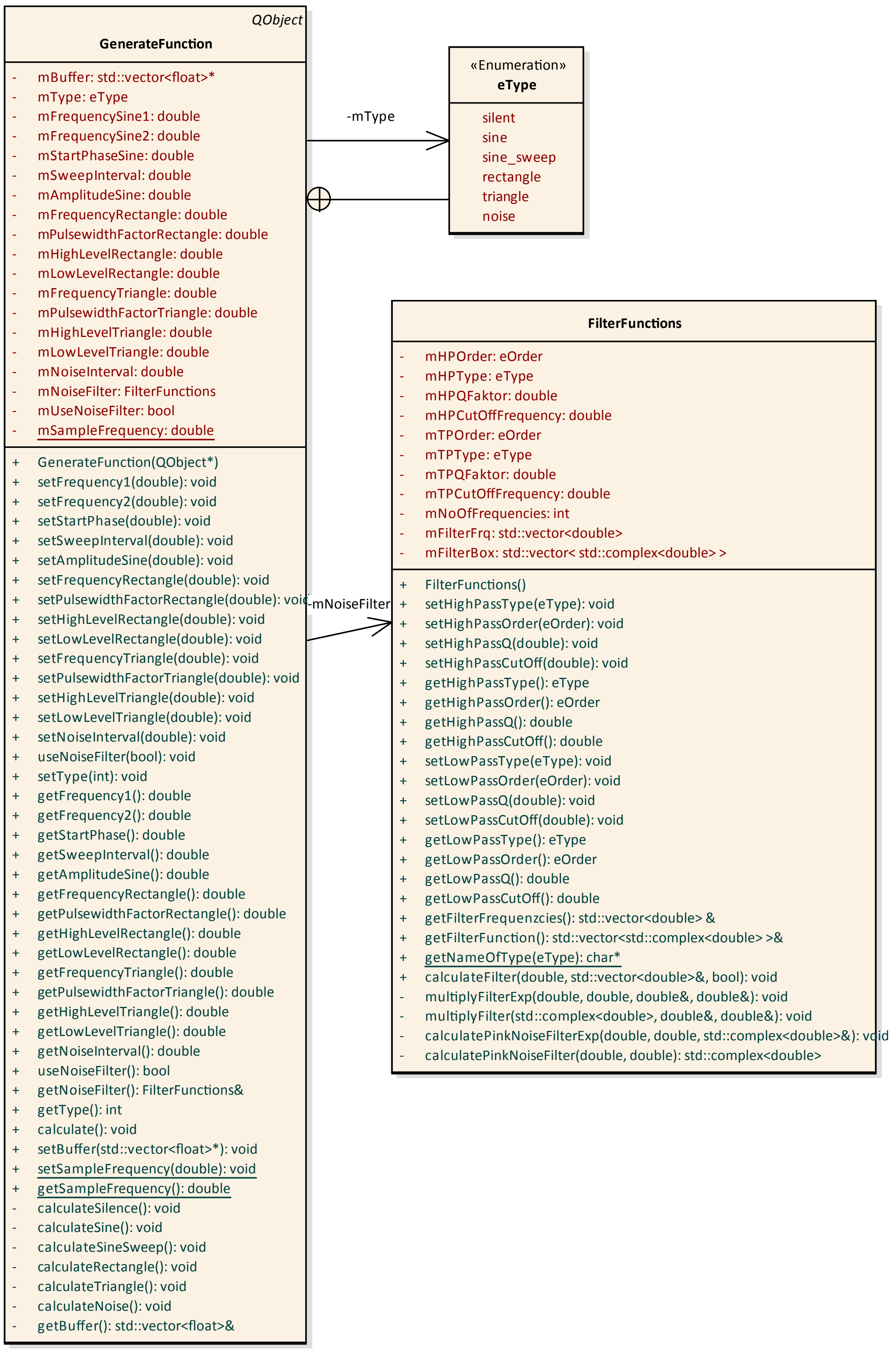
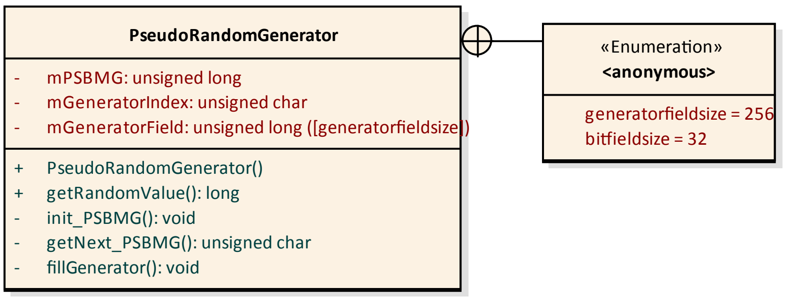


```
QObject
GenerateFunction
-  mBuffer: std::vector<float>*
-  mType: eType
-  mFrequencySine1: double
-  mFrequencySine2: double
-  mStartPhaseSine: double
-  mSweepInterval: double
-  mAmplitudeSine: double
-  mFrequencyRectangle: double
-  mPulsewidthFactorRectangle: double
-  mHighLevelRectangle: double
-  mLowLevelRectangle: double
-  mFrequencyTriangle: double
-  mPulsewidthFactorTriangle: double
-  mHighLevelTriangle: double
-  mLowLevelTriangle: double
-  mNoiseInterval: double
-  mNoiseFilter: FilterFunctions
-  mUseNoiseFilter: bool
-  mSampleFrequency: double
+  GenerateFunction(QObject*)
+  setFrequency1(double): void
+  setFrequency2(double): void
+  setStartPhase(double): void
+  setSweepInterval(double): void
+  setAmplitudeSine(double): void
+  setFrequencyRectangle(double): void
+  setPulsewidthFactorRectangle(double): void
+  setHighLevelRectangle(double): void
+  setLowLevelRectangle(double): void
+  setFrequencyTriangle(double): void
+  setPulsewidthFactorTriangle(double): void
+  setHighLevelTriangle(double): void
+  setLowLevelTriangle(double): void
+  setNoiseInterval(double): void
+  useNoiseFilter(bool): void
+  setType(int): void
+  getFrequency1(): double
+  getFrequency2(): double
+  getStartPhase(): double
+  getSweepInterval(): double
+  getAmplitudeSine(): double
+  getFrequencyRectangle(): double
+  getPulsewidthFactorRectangle(): double
+  getHighLevelRectangle(): double
+  getLowLevelRectangle(): double
+  getFrequencyTriangle(): double
+  getPulsewidthFactorTriangle(): double
+  getHighLevelTriangle(): double
+  getLowLevelTriangle(): double
+  getNoiseInterval(): double
+  useNoiseFilter(): bool
+  getNoiseFilter(): FilterFunctions&
+  getType(): int
+  calculate(): void
+  setBuffer(std::vector<float>*): void
+  setSampleFrequency(double): void
+  getSampleFrequency(): double
-  calculateSilence(): void
-  calculateSine(): void
-  calculateSineSweep(): void
-  calculateRectangle(): void
-  calculateTriangle(): void
-  calculateNoise(): void
-  getBuffer(): std::vector<float>&
```

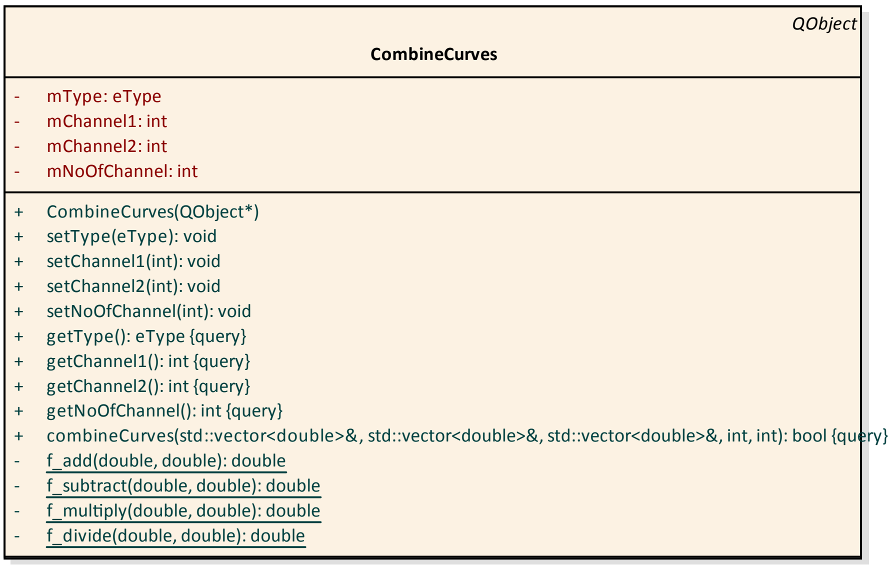
```
«Enumeration»
eType
   silent
   sine
   sine_sweep
   rectangle
   triangle
   noise
```

```
FilterFunctions
-  mHPOrder: eOrder
-  mHPType: eType
-  mHPQFaktor: double
-  mHPCutOffFrequency: double
-  mTPOrder: eOrder
-  mTPType: eType
-  mTPQFaktor: double
-  mTPCutOffFrequency: double
-  mNoOfFrequencies: int
-  mFilterFrq: std::vector<double>
-  mFilterBox: std::vector< std::complex<double> >
+  FilterFunctions()
+  setHighPassType(eType): void
+  setHighPassOrder(eOrder): void
+  setHighPassQ(double): void
+  setHighPassCutOff(double): void
+  getHighPassType(): eType
+  getHighPassOrder(): eOrder
+  getHighPassQ(): double
+  getHighPassCutOff(): double
+  setLowPassType(eType): void
+  setLowPassOrder(eOrder): void
+  setLowPassQ(double): void
+  setLowPassCutOff(double): void
+  getLowPassType(): eType
+  getLowPassOrder(): eOrder
+  getLowPassQ(): double
+  getLowPassCutOff(): double
+  getFilterFrequenzcies(): std::vector<double> &
+  getFilterFunction(): std::vector<std::complex<double> >&
+  getNameOfType(eType): char*
+  calculateFilter(double, std::vector<double>&, bool): void
-  multiplyFilterExp(double, double, double&, double&): void
-  multiplyFilter(std::complex<double>, double&, double&): void
-  calculatePinkNoiseFilterExp(double, double, std::complex<double>&): void
-  calculatePinkNoiseFilter(double, double): std::complex<double>
```



```
PseudoRandomGenerator
-  mPSBMG: unsigned long
-  mGeneratorIndex: unsigned char
-  mGeneratorField: unsigned long ([generatorfieldsize])
+  PseudoRandomGenerator()
+  getRandomValue(): long
-  init_PSBMG(): void
-  getNext_PSBMG(): unsigned char
-  fillGenerator(): void
```

```
«Enumeration»
<anonymous>
   generatorfieldsize = 256
   bitfieldsize = 32
```



```
QObject
CombineCurves
-  mType: eType
-  mChannel1: int
-  mChannel2: int
-  mNoOfChannel: int
+  CombineCurves(QObject*)
+  setType(eType): void
+  setChannel1(int): void
+  setChannel2(int): void
+  setNoOfChannel(int): void
+  getType(): eType {query}
+  getChannel1(): int {query}
+  getChannel2(): int {query}
+  getNoOfChannel(): int {query}
+  combineCurves(std::vector<double>&, std::vector<double>&, std::vector<double>&, int, int): bool {query}
-  f_add(double, double): double
-  f_subtract(double, double): double
-  f_multiply(double, double): double
-  f_divide(double, double): double
```