```
Start ::=
   { OptionStatement }
   { ImportsStatement}
   [ Attributes ]
   { NamespaceMemberDeclaration}

Modifier ::= AccessModifier| "Shadows" | "MustInherit" | "NotInheritable"|
  "Shared" | "Overridable" | "NotOverridable" | "MustOverride" |
  "Overrides" | "Overloads"| "ReadOnly" |"WithEvents"| "Default" | "WriteOnly"
AtMod ::= [ Attributes ] {Modifier}

AccessModifier ::= "Public" | "Protected" | "Friend" | "Private"

QualifiedIdentifier ::= "ID" |  QualifiedIdentifier "." "ID"
NamespaceOrTypeName ::= QualifiedIdentifier

Attributes ::= "<" AttributeList ">"

AttributeList ::= Attribute |AttributeList "," Attribute
Attribute ::=
   [ AttributeModifier ":" ] TypeName1 [ "(" [ AttributeArguments ] ")" ]
AttributeModifier ::= "Assembly" | "Module"

AttributeArguments ::=
   AttributePositionalArgumentList |
   AttributePositionalArgumentList "," VariablePropertyInitializerList |
   VariablePropertyInitializerList
AttributePositionalArgumentList ::=
   ConstantExpression | AttributePositionalArgumentList "," ConstantExpression
VariablePropertyInitializerList ::=
   VariablePropertyInitializer |
   VariablePropertyInitializerList "," VariablePropertyInitializer
VariablePropertyInitializer ::=  "ID" ":=" ConstantExpression

OptionStatement ::=
   OptionExplicitStatement | OptionStrictStatement | OptionCompareStatement

OptionExplicitStatement ::= "Option" "Explicit" [ OnOff ] "NL"
OnOff ::= "On" | "Off"

OptionStrictStatement ::= "Option" "Strict" [ OnOff ] "NL"

OptionCompareStatement ::= "Option" "Compare" CompareOption "NL"
CompareOption ::= "Binary" | "Text"

ImportsStatement ::= "Imports" ImportsClauses "NL"
ImportsClauses ::= ImportsClause | ImportsClauses "," ImportsClause
ImportsClause ::= ImportsAliasClause | ImportsNamespaceClause

ImportsAliasClause ::= "ID" "=" NamespaceOrTypeName

ImportsNamespaceClause ::= NamespaceOrTypeName

NamespaceDeclaration ::=
   "Namespace" QualifiedIdentifier "NL"
   { NamespaceMemberDeclaration }
   "End" "Namespace" "NL"

NamespaceMemberDeclaration ::=  NamespaceDeclaration | TypeDeclaration
TypeDeclaration ::= ModuleDeclaration | NonModuleDeclaration
NonModuleDeclaration ::=
   EnumDeclaration |  StructureDeclaration |  InterfaceDeclaration |
   ClassDeclaration |  DelegateDeclaration

TypeName ::= TypeName1 | ArrayTypeName
TypeName1 ::= NamespaceOrTypeName | BuiltInTypeName
BuiltInTypeName ::= "Object" | PrimitiveTypeName
```

```
TypeImplementsClause ::= "Implements" Implements "NL"
Implements ::=   TypeName | Implements "," TypeName

PrimitiveTypeName ::= NumericTypeName | "Boolean" | "Date" | "Char" | "String"
NumericTypeName ::= IntegralTypeName | FloatingPointTypeName | "Decimal"
IntegralTypeName ::= "Byte" | "Short" | "Integer" | "Long"
FloatingPointTypeName ::= "Single" | "Double"

EnumDeclaration ::=
   AtMod "Enum" "ID" [ "As" IntegralTypeName ] "NL"
   { EnumMemberDeclaration }
   "End" "Enum" "NL"

EnumMemberDeclaration ::=
   [ Attributes ] "ID" [ "=" ConstantExpression ] "NL"

ClassDeclaration ::=
   AtMod "Class" "ID" "NL"
   [ ClassBase ]
   { TypeImplementsClause }
   { ClassMemberDeclaration }
   "End" "Class" "NL"

ClassBase ::= "Inherits" TypeName "NL"

ClassMemberDeclaration ::=
   NonModuleDeclaration | EventMemberDeclaration | VariableMemberDeclaration |
   ConstantMemberDeclaration |  MethodMemberDeclaration |
   PropertyMemberDeclaration |  ConstructorMemberDeclaration

StructureDeclaration ::=
   AtMod "Structure" "ID" "NL"
   { TypeImplementsClause }
   { StructMemberDeclaration }
   "End" "Structure" "NL"

StructMemberDeclaration ::=
   NonModuleDeclaration |  VariableMemberDeclaration |
   ConstantMemberDeclaration | EventMemberDeclaration |
   MethodMemberDeclaration | PropertyMemberDeclaration |
   ConstructorMemberDeclaration

ModuleDeclaration ::=
   AtMod "Module" "ID" "NL"
   { ModuleMemberDeclaration }
   "End" "Module" "NL"

ModuleMemberDeclaration ::=
   NonModuleDeclaration | VariableMemberDeclaration |
   ConstantMemberDeclaration |  EventMemberDeclaration |
   MethodMemberDeclaration | PropertyMemberDeclaration |
   ConstructorMemberDeclaration

InterfaceDeclaration ::=
   AtMod "Interface" "ID" "NL"
   { InterfaceBase }
   { InterfaceMemberDeclaration }
   "End" "Interface" "NL"


InterfaceBase ::= "Inherits" InterfaceBases "NL"
InterfaceBases ::=  TypeName |  InterfaceBases "," TypeName

InterfaceMemberDeclaration ::=
   NonModuleDeclaration |  EventMemberDeclaration |
   MethodMemberDeclaration |  PropertyMemberDeclaration

ArrayTypeName ::= TypeName1 ArrayTypeModifiers
```

```
ArrayTypeModifiers ::= ArrayTypeModifier {ArrayTypeModifier}
ArrayTypeModifier ::= "(" ArgumentList ")"
ArrayNameModifier ::= ArraySizeInitializationModifier

DelegateDeclaration ::= AtMod "Delegate" MethodDeclaration
ImplementsClause ::= "Implements" ImplementsList
ImplementsList ::=
   InterfaceMemberSpecifier | ImplementsList "," InterfaceMemberSpecifier
InterfaceMemberSpecifier ::= TypeName "." "ID"

MethodMemberDeclaration ::= MethodDeclaration | ExternalMethodDeclaration

MethodDeclaration ::= SubDeclaration | FunctionDeclaration
SubDeclaration ::=
   AtMod "Sub" "ID" [ "(" [ParameterList] ")" ] [ HandlesOrImplements ] "NL"
   Block
   "End" "Sub" "NL"
FunctionDeclaration ::=
   AtMod  "Function" "ID" [ "(" [ParameterList] ")" ]
   [ "As" [ Attributes ] TypeName ] [ HandlesOrImplements ] "NL"
   Block
   "End" "Function" "NL"

HandlesOrImplements ::= HandlesClause | ImplementsClause

ExternalMethodDeclaration ::=
   ExternalSubDeclaration | ExternalFunctionDeclaration
ExternalSubDeclaration ::=
   AtMod "Declare" [ CharsetModifier ] "Sub" "ID" LibraryClause
   [ AliasClause ] [ "(" [ ParameterList ] ")" ] "NL"
ExternalFunctionDeclaration ::=
   AtMod "Declare" [ CharsetModifier ] "Function" "ID" LibraryClause
   [ AliasClause ] [ "(" [ ParameterList ] ")" ]
   [ "As" [ Attributes ] TypeName ] "NL"

CharsetModifier ::= "Ansi" | "Unicode" | "Auto"
LibraryClause ::= "Lib" "STR"
AliasClause ::= "Alias" "STR"

ParameterList ::=  Parameter |  ParameterList "," Parameter
Parameter ::=
   [ Attributes ] ParameterModifier {ParameterModifier} "ID" [ "As" TypeName ]
   [ "=" ConstantExpression ]
ParameterModifier ::= "ByVal" | "ByRef" | "Optional" | "ParamArray"

HandlesClause ::= "Handles" EventHandlesList
EventHandlesList ::=
   EventMemberSpecifier | EventHandlesList "," EventMemberSpecifier
EventMemberSpecifier ::=  "ID" "." "ID" | "MyBase" "." "ID"

ConstructorMemberDeclaration ::=
   AtMod "Sub" "New" [ "(" [ParameterList] ")" ] "NL"
   Block
   "End" "Sub" "NL"

EventMemberDeclaration ::=
   AtMod "Event" "ID" ParametersOrType [ ImplementsClause ]
ParametersOrType ::=  [ "(" [ ParameterList ] ")" ] | "As" TypeName

ConstantMemberDeclaration ::=  AtMod "Const" ConstantDeclarators "NL"

ConstantDeclarators ::=
   ConstantDeclarator |  ConstantDeclarators "," ConstantDeclarator
ConstantDeclarator ::= "ID" [ "As" TypeName ] "=" ConstantExpression "NL"

VariableMemberDeclaration ::=  AtMod [ "Dim" ] VariableDeclarators "NL"

VariableDeclarators ::=
```

```
   VariableDeclarator | VariableDeclarators "," VariableDeclarator
VariableDeclarator ::=
   VariableIdentifier [ "As" [ "New" ] TypeName1 [ArrayTypeModifier] ]
   [ "=" VariableInitializer ]
VariableIdentifiers ::=
   VariableIdentifier | VariableIdentifiers "," VariableIdentifier
VariableIdentifier ::= "ID" [ ArrayNameModifier ]

VariableInitializer ::= RegularInitializer | ArrayElementInitializer

RegularInitializer ::= Expression

ArraySizeInitializationModifier ::=  ArrayTypeModifiers [ ArrayTypeModifiers ]
UpperBoundList::=  Expression | UpperBoundList "," Expression

ArrayElementInitializer ::= "{" [ VariableInitializerList ] "}"
VariableInitializerList ::=
   VariableInitializer | VariableInitializerList "," VariableInitializer

PropertyMemberDeclaration ::=
   AtMod "Property" "ID"
   [ "(" [ ParameterList ] ")" ] [ "As" TypeName ] [ ImplementsClause ] "NL"

   { PropertyAccessorDeclaration }
   "End" "Property" "NL"
PropertyAccessorDeclaration ::=
   PropertyGetDeclaration | PropertySetDeclaration

PropertyGetDeclaration ::=
   [ Attributes ] "Get" "NL"
   Block
   "End" "Get" "NL"

PropertySetDeclaration ::=
   [ Attributes ] "Set" [ "(" ParameterList ")" ] "NL"
   Block
   "End" "Set" "NL"

Statement ::=
   LocalDeclarationStatement | WithStatement | SyncLockStatement |
   EventStatement | AssignmentStatement | InvocationStatement |
   ConditionalStatement | LoopStatement | ErrorHandlingStatement |
   BranchStatement | ArrayHandlingStatement
StatementTerminator ::= "NL" | ":"

Block ::= { LabeledLine }
LabeledLine ::= [ LabelName ":" ] Statements "NL"
LabelName ::= "ID" | "INT"
Statements ::= Statement | Statements ":" Statement

LocalDeclarationStatement ::=  LocalModifier LocalDeclarators
LocalModifier ::= "Static" | "Dim" | "Const"
LocalDeclarators ::=
   LocalDeclarator | LocalDeclarators "," LocalDeclarator
LocalDeclarator ::=
   LocalIdentifier [ "As" [ "New" ] TypeName1 [ArrayTypeModifier] ]
   [ "=" VariableInitializer ]
LocalIdentifiers ::=
   LocalIdentifier | LocalIdentifiers "," LocalIdentifier
LocalIdentifier ::= "ID" [ ArrayNameModifier ]

WithStatement ::=
   "With" Expression StatementTerminator
   Block
   "End" "With"

SyncLockStatement ::=
   "SyncLock" Expression StatementTerminator
```

```
   Block
   "End" "SyncLock"

EventStatement ::=
   RaiseEventStatement |  AddHandlerStatement |  RemoveHandlerStatement

RaiseEventStatement ::= "RaiseEvent" SimpleNameExpression [ArrayTypeModifier]

AddHandlerStatement ::=  "AddHandler" Expression "," Expression
RemoveHandlerStatement ::= "RemoveHandler" Expression "," Expression

AssignmentStatement ::=
   RegularAssignmentStatement | CompoundAssignmentStatement |
   MidAssignmentStatement

RegularAssignmentStatement ::=  MatExpression9 [ "=" Expression ]

CompoundAssignmentStatement ::=
   MatExpression9 CompoundBinaryOperator "=" Expression
CompoundBinaryOperator ::=
   "^" | "*" | "/" | "\" | "+" | "-" | "&" | "<<" | ">>"

MidAssignmentStatement ::=
   "Mid" ( Expression "," Expression [ "," Expression ] ) "=" Expression

InvocationStatement ::= "Call" InvocationExpression

ConditionalStatement ::= IfStatement | SelectStatement

IfStatement ::= BlockIfStatement  | LineIfThenStatement
BlockIfStatement ::=
   "If" Expression [ "Then" ] StatementTerminator
   Block
   { ElseIfStatement }
   [ ElseStatement ]
   "End" "If"
ElseIfStatement ::=
   "ElseIf" Expression [ "Then" ] StatementTerminator
   Block
ElseStatement ::=
   "Else" StatementTerminator
   Block

LineIfThenStatement ::=
   "If" Expression "Then"
   ( Statements [ "Else" Statements ] | "Else" Statements )

SelectStatement ::=
   "Select" [ "Case" ] Expression StatementTerminator
   { CaseStatement }
   [ CaseElseStatement ]
   "End" "Select"
CaseStatement ::=
   "Case" CaseClauses StatementTerminator
   Block
CaseClauses ::=  CaseClause | CaseClauses "," CaseClause
CaseClause ::=
   ["Is"] ComparisonOperator Expression | Expression [ "To" Expression ]
ComparisonOperator ::= "=" | "<>" | "<" | ">" | ">=" | "<="
CaseElseStatement ::=
   "Case" "Else" StatementTerminator
   Block

LoopStatement ::=
   WhileStatement | DoLoopStatement | ForStatement | ForEachStatement

WhileStatement ::=
   "While" Expression StatementTerminator
```

```
   Block
   "End" "While"
DoLoopStatement ::=
   "Do" [ WhileOrUntil Expression ] StatementTerminator
   Block
   "Loop" [ WhileOrUntil Expression ]
WhileOrUntil ::= "While" | "Until"

ForStatement ::=
   "For" LoopControlVariable "=" Expression "To" Expression [ "Step" Expression
]
      StatementTerminator
   Block
   "Next" [ NextExpressionList ]
LoopControlVariable ::= "ID" "As" TypeName | Expression
NextExpressionList ::=  Expression | NextExpressionList "," Expression

ForEachStatement ::=
   "For" "Each" LoopControlVariable "In" Expression StatementTerminator
   Block
   "Next" [Expression ]

ErrorHandlingStatement ::=
   StructuredErrorStatement | UnstructuredErrorStatement

StructuredErrorStatement ::=  ThrowStatement | TryStatement
TryStatement ::=
   "Try" StatementTerminator
   Block
   { CatchStatement }
   [ FinallyStatement ]
   "End" "Try"

FinallyStatement ::=
   "Finally" StatementTerminator
   Block

CatchStatement ::=
   "Catch" [ "ID" "As" TypeName ] [ "When" Expression ]StatementTerminator
   Block

ThrowStatement ::= "Throw" [ Expression ]

UnstructuredErrorStatement ::=
   ErrorStatement | OnErrorStatement | ResumeStatement

ErrorStatement ::= "Error" Expression

OnErrorStatement ::= "On" "Error" ErrorClause
ErrorClause ::=  "GoTo" "-" "INT" | GotoStatement | "Resume" "Next"
ResumeStatement ::= "Resume" [ ResumeClause ]
ResumeClause ::= "Next" | LabelName

BranchStatement ::=
   GotoStatement | ExitStatement | StopStatement | ReturnStatement
GotoStatement ::= "GoTo" LabelName
ExitStatement ::= "Exit" ExitKind
ExitKind ::=
   "Do" | "For" | "While" | "Select" | "Sub" | "Function" |"Property" | "Try"
StopStatement ::= "Stop"
ReturnStatement ::= "Return" [ Expression ]

ArrayHandlingStatement ::=  RedimStatement | EraseStatement

RedimStatement ::= "ReDim" [ "Preserve" ] RedimClauses
RedimClauses ::= RedimClause |  RedimClauses "," RedimClause
RedimClause ::= Expression ArraySizeInitializationModifier
```

```
EraseStatement ::= "Erase" EraseExpressions
EraseExpressions ::=  Expression | EraseExpressions "," Expression


ConstantExpression ::= Expression
Expression ::= LogExpression2 { "Xor" LogExpression2 }
LogExpression2 ::= LogExpression3 { ( "Or" | "OrElse" ) LogExpression3 }
LogExpression3 ::= LogExpression4 { ( "And" | "AndAlso" ) LogExpression4 }
LogExpression4 ::= { "Not" } IsExpression
IsExpression ::=
   "TypeOf" CompExpression "Is" TypeName |
   CompExpression [ "Is" CompExpression ]
CompExpression ::= MatExpression { Relation MatExpression }
Relation ::= "=" | "<" | ">" | "<>" | "<=" | ">=" | "Like"
MatExpression ::= MatExpression2 { "&" MatExpression2 }
MatExpression2 ::= MatExpression3 { ("+" | "-") MatExpression3 }
MatExpression3 ::= MatExpression4 { "Mod" MatExpression4 }
MatExpression4 ::= MatExpression5 { "\" MatExpression5 }
MatExpression5 ::= MatExpression6 { ("*" | "/") MatExpression6 }
MatExpression6 ::= MatExpression7 { ("<<" | ">>") MatExpression7 }
MatExpression7 ::= {"+" | "-"} MatExpression8
MatExpression8 ::= MatExpression9 ["^" MatExpression8]
MatExpression9 ::= { "AddressOf" } InvocationExpression
InvocationExpression ::= MemberAccessExpression [ ArrayTypeModifier ]

MemberAccessExpression ::=
   MemberAccessBase "." "ID" | "." "ID" | DictionaryAccessExpression
MemberAccessBase ::= InvocationExpression | BuiltInTypeName
DictionaryAccessExpression ::=  DictionaryAccessExpression "!" "ID" | "!" "ID" |
SimpleExpression

SimpleExpression ::=
   LiteralExpression  | ParenthesizedExpression | InstanceExpression |
   SimpleNameExpression | GetTypeExpression | NewExpression | CastExpression

LiteralExpression ::= Literal
ParenthesizedExpression ::= "(" Expression ")"
InstanceExpression ::= "Me" | "MyClass" | "MyBase"
SimpleNameExpression ::= "ID"
GetTypeExpression ::= "GetType" "(" TypeName ")"

ArgumentList ::=
   PositionalArgumentList "," NamedArgumentList |
   PositionalArgumentList | NamedArgumentList
PositionalArgumentList ::=
   [ Expression ] | PositionalArgumentList "," [ Expression ]
NamedArgumentList ::=
   "ID" ":=" Expression | NamedArgumentList "," "ID" ":=" Expression

NewExpression ::=
   ObjectCreationExpression | ArrayCreationExpression |
   DelegateCreationExpression

ObjectCreationExpression ::= "New" TypeName1 [ ArrayTypeModifier ]

ArrayCreationExpression ::=
   "New" TypeName [ ArraySizeInitializationModifier ] ArrayElementInitializer

DelegateCreationExpression ::= "New" TypeName "(" Expression ")"

CastExpression ::=
   "DirectCast" "(" Expression "," TypeName ")" |
   "CType" "(" Expression "," TypeName ")" |
   CastTarget "(" Expression ")"
CastTarget ::=
   "CBool" | "CByte" | "CChar" | "CDate" | "CDec" | "CDbl" |
   "CInt" | "CLng" | "CObj" | "CShort" | "CSng" | "CStr"
```

```
Literal ::=
   "True" | "False" | "Nothing" | "INT" | "FLT" | "STR" | "CHR" | "DTM"
```