# C♭ AST Format

The provided code implements the AST representation described below.

## Data Types

The `AST` type is an abstract class which provides a tag and a source code line number. The tag identifies the kind of node; the line number is stored for use in error messages.

There are three subclasses of the `AST` class:

- `AST_nonleaf` used for node types which have a fixed number of children (but at least one).
- `AST_kary` used for node types which have a variable number of children, from zero up.
- `AST_leaf` used for node types which can *never* have any children.

Static factory methods in the AST base class are normally used to instantiate AST nodes. These are as follows.

- `AST.NonLeaf( NodeType tag, int ln, ... )`
- `AST.AST_kary Kary( NodeType tag, int ln, ... )`
- `AST.Leaf( NodeType tag, int ln, string s )`
- `AST.Leaf( NodeType tag, int ln, int i )`
- `AST.Leaf( NodeType tag, int ln )`

In each case, the first two parameters are the tag and the source code line number to be stored in the node. For the `NonLeaf` method, there are one or more additional arguments of type `AST` which are references to the children of this new node. The children (i.e. subtrees) must have already been constructed. The `AST.Kary` method constructs a new node of the `AST_kary` type. There are normally zero children when the node is created ... each subtree is constructed later and added as a child of this node by invoking its `AddChild` method. However, if there are some initial children already constructed, they can be provided as additional arguments to the `Kary` method.

There are three versions of the `AST.Leaf` method. The one with a `string` argument is used when the node has an associated `string` value (such as an identifier or a string constant). The one with an extra `int` argument is used when the node has as associated `int` value (such as an `int` constant). The version with no additional argument is used when the leaf node has neither an associated `int` or `string` value (such as the value `null`).

## AST Node Tag Type

The tags are implemented as values of an *enum* type named `NodeType`. Its constants have the following names.

```
Program, UsingList, ClassList, Class, MemberList, Const, Field, Method,
IdList, FormalList, Formal, Static, Virtual, Override, Array, Block,
LocalDecl, Assign, If, While, Break, Return, Call, ActualList,
PlusPlus, MinusMinus, Empty, Add, Sub, Mul, Div, Mod, And, Or, Equals,
NotEquals, LessThan, GreaterThan, LessOrEqual, GreaterOrEqual,
```

`UnaryMinus, UnaryPlus, Index, NewArray, NewClass, Null, Cast, Dot, IntConst, StringConst, CharConst, Ident`

## AST Node Types

If the `Arity` entry shows 0, then the `Tag` is used for a leaf node type (`AST_leaf`). If it appears as *k*, then the `Tag` is used for a *k*-ary node type (AST_kary); otherwise the tag is used for a non-leaf node with a fixed number of children (`AST_nonleaf`).

| Tag | Arity | Description |
|---|---|---|
| `Program` | 2 | Used at root of AST. Children are a using-list, and a list of class declarations. |
| `UsingList` | k | Children are 0 or more identifiers which were declared after the keyword `using` |
| `ClassList` | k | Children are 1 or more class declarations |
| `Class` | 3 | Children are the class name (an identifier), the parent class (an identifier), and a list of member declarations. If the parent class is omitted in the source code, `null` should be used. |
| `MemberList` | k | Children are 0 or more const, field or method declarations |
| `Const` | 3 | A constant declaration; children are the type, the identifier and the value |
| `Field` | 2 | Children are the type and a list of identifiers |
| `Method` | 5 | A method declaration; children are the result type, the method name identifier, a list of formal parameters, the method body (a block), and method usage attribute. If the method has the `void` type, the first child is missing, i.e. the child reference is `null`. The last child distinguishes static/virtual/override. |
| `IdList` | k | Children are 0 or more identifiers |
| `FormalList` | k | Children are 0 or more formal parameter declarations |
| `Formal` | 2 | Children are a type and an identifier (the name of the formal). |
| `Static`<br>`Virtual`<br>`Override` | 0 | Specified where a method is declared as static, virtual or override, respectively. |
| `Array` | 1 | Represents an array type; the child is the element type |
| `Block` | k | Children are 0 or more local declarations or statements |
| `LocalDecl` | 2 | Children are a type and a list of identifiers |
| `Assign` | 2 | An assignment statement; children are the LHS and the RHS |

| Tag | Arity | Description |
|---|---|---|
| `If` | 3 | An if statement; children are the test expression, the then-part statement and the else-part statement (which is never omitted) |
| `While` | 2 | A while statement; children are the test expression and the statement forming the body. |
| `Break` | 0 | A break statement |
| `Return` | 1 | A return statement; the child is the expression to be returned if there is one, otherwise the child reference is `null` |
| `Call` | 2 | A method call; children are the method name and the arguments (actual parameters); it can be used as either an expression or a statement |
| `ActualList` | k | Children are 0 or expressions (to be used as actual parameters). |
| `PlusPlus` `MinusMinus` | 1 | The child is the operand of a postfix `++` or `--` operator. |
| `Empty` | 0 | This is an empty statement. It must also be generated when the *else* part of an *if-statement* is omitted |
| `Add Sub Mul Div Mod` | 2 | An expression where the operator is the binary `+ - * / %` |
| `And Or` | 2 | An expression where the operator is `&&` or `\|\|` |
| `Equals` `NotEquals` `LessThan` `GreaterThan` `LessOrEqual` `GreaterOrEqual` | 2 | An expression where the operator is `== != < > <= >=` |
| `UnaryMinus` `UnaryPlus` | 1 | An expression where the operator is a prefix `-` or prefix `+` |
| `Index` | 2 | The children are an expression (which is an array or string value) and an index expression |
| `NewArray` | 2 | The children are the element type of the array and the size |
| `NewClass` | 1 | The child is an identifier providing the name of a class type |
| `Null` | 0 | The null reference |
| `Cast` | 2 | The children are intended to be a type and an expression; however note that the grammar may accept constructions for a type which are invalid – subsequent checking must verify validity. |

| Tag | Arity | Description |
|---|---|---|
| `Dot` | 2 | The LHS is an expression and the RHS is an identifier |
| `IntConst`<br>`StringConst`<br>`CharConst` | 0 | Constants |
| `Ident` | 0 | An identifier (which could represent a class, constant, field, method, local variable or the 'Length' property) |

| | | |
|---|---|---|
| IntType | 0 | A builtin type |
| StringType | 0 | A builtin type |
| CharType | 0 | A builtin type |