Namespace BOOSE

Classes

AboutBOOSE

Array

An array of integers or reals. array int myArray 10,2\narray real prices 10\n[array][(type)][(size)][(optional dimentions, 1 if not given)]

BOOSEException

Generic BOOSE Language exception Extends Exception

Boolean

Boolean datatype, also used for Conditional Commands.

Call

Command to call a previously defined method. method int testMethod\n int one, int two\n testMethod = one* two\nend method\nint global = 55\ncall testMethod 10 15

Canvas

Abstract class that implements CanvasInterface. See CanvasInterface documentation for what to implement. This class handles all the drawing on your system according to above documentation. BOOSE does not by default specify a default drawing Canvas (only this).

CanvasCommand

Abstract class that is inherited by derived canvas commands that draw on a canvas. It stores the x and y positions of the last drawing operation.

CanvasException

Exception that should be thrown by operations that fail on the Canvas.

Cast

Cast real values to int values. Syntax: cast *intVarrealVar* Both variables must be previously defined. real circ = 100.123\nint c\ncast c circ\ncircle c

Circle

Draw a circle round the current cursor position. circle 50Canvas exception if a non-integer is passed.

Command

Abstract class for BOOSE commands. Can handle a parameter list or expression. This class will seperate the parameters or expression from the command and store in the "ParameterList". The Command's Compile() method is called by the parser and this is expected to set the Command's parameters.

BOOSE implementation of commands makes definitions "sealed". Sealed classes cannot be inherited, so you can't use what's there to remove the restrictions, you will have to totally rewrite them.

CommandException

Exception genereted by the StoredProgram class when a command fails

CommandFactory

CommandOneParameter

Commands with one parameter. The command system adds one parameter at a time. Can be used by any one parameter command.

CommandThreeParameters

Commands with two parameters. Each command has an Xpos and Ypos that operates from the current cursor position.

CommandTwoParameters

Commands with two parameters. Each command has an Xpos and Ypos that operates from the current cursor position.

CompoundCommand

CompoundCommand is a building block for shared functionality between CompoundCommands, such as IF, WHILE, FOR ,ELSE and ENDS Classes the inherit from here tend to be "sealed" so that they cnnot be inherited from to manipulate restrictions.

ConditionalCommand

Used for Ifs, Whiles and Fors

DrawTo

Draw a line from to current cursor position to the provided x,y position. drawto 200,200

Else

An Else is like an Endif but it pops the corresponding If off the stack and pushes itself on so that it becomes the corresponding command to the end

End

End command for If/While/For/Method. BOOSE has only one End type command which has a parameter to say what it belongs so eg. "end if". End checks Boose types for the ciorresponding command (If/While/For/Method) and will likely require a complete rewrite to replcae those commands. It would probbaly be easier to replace Parser, StoredProgram, CompoundCommand and this. Look at the metghods each provides, if you like, and try and fill in their fucntionality.

Evaluation

An evaluation encompasses all things that can have a value, such as variables, expressions and conditions used in loops and ifs.

FactoryException

exception genereted by the StoredProgram class

For

For command. Adds processing at compile() and execute() to process a for command.

If

If command. Adds processing at compile() and execute() to process an if command.

Int

Command Class to define and store integer values.

Method

Method declaration command. Overcoming method restrictions will require a complete rewrite. It would be simpler to do this after Parser and StoredProgram. You are getting to the stage where you need to do a big rewrite.

MoveTo

Set the cursor position, for subsequent drawing operations, to the provide x,y position.

Parser

The parser takes a program, who's lines are seperated by "\n", and attempts to create a StoredProgram. Once the SToredProgram has been created it then can be run. As much processing that can be done is done here because it has no run penalty (other than starting the program). Command objects are created and their parameters processed as much as possible. Variables are checked for existance but expressions cannot be processed until runtime.

ParserException

exception genereted by the Parser class to denote syntax errors

Peek

Peek an array, i.e. get the value of an array cell. [variable] = peek [array name] [row] [optional column]

PenColour

Set the pen for the next drawing operations to the provided r,g,b colour.

Poke

Poke an array, i.e. set the value of an array cell. poke [array name] [row] [optional column] = [value]

Real

Class for a Real Number Variable.

Rect

Draw a rectangle of provided with and heigh with the current cursor position being the top left corner.

RestrictionException

Exception thrown when restrictions are breached. If you get this you need to write your own code for the facility.

StoredProgram

A collection class for storing a program of Command objects, extends ArrayList to add a program counter and a flag to indicate that the syntax is ok and the program valid. Adds methods below to process variables and methods and implements flow of control for ifs and loops.

StoredProgramException

exception genereted by the StoredProgram class

VarException

Exception raised by variable commands.

While

While command. So far blank as it doesn't do anything beyond ConditionalCommand but it makes the code clearer (in the factory).

Write

Interfaces

ICanvas

Implement ICanvas for your BOOSE renderer. It has an Xpos and Ypos of the current cursor position, and a pen colour. Your class should implement the methods below to draw on its "bitmap" (i.e. it may not be a bitmap, it could draw in ASCII text for example).

ICommand

Interface for new commands. Any new Command class should implement this interface and be generated by a CommandFactory that implements the ICommandFactory interface. Contains methods to set the Command up after its creation, manage its parameters when it is compiled and execute it when it is called.

ICommandFactory

To add commands to BOOSE you must create a CommandFactory that uses this interface. You should extend the existing BOOSE:CommandFactory (which implements this interface) and then implement the MakeCommand() method. It shoudld create a new command object based on the string passed. Any standard BOOSE commands can then be made by calling base.MakeCommand();

IEvaluation

IEvaluation adds properties to get and set an Evaluations name, value and expression. An evaluation can be a variable declaration, such as "int total" or "real area". It can be change of value of a variable, such as "total = total + 1", the part after the "=" is an referred to as an "expression". In the first two cases the parser will generate an Int object and a Real Object. In the third case it will generate a "Evaluation" object.

IParser

The Parser Class takes a BOOSE program as a String with each command seperated by '\n' and creates command objects for each valid command and stores them in the passed in StoredProgram. Exceptions are generated for any syntax errors. When each valid command is generated its Compile() method is called. The valid command will have its parameters processed and any variables identified. It is the role of StoredProgram to run the commands.

IStoredProgram

Enums

ConditionalCommand.conditionalTypes

Enumerated type to represent individual conditional commands.

Class AboutBOOSE

Namespace: BOOSE

Assembly: BOOSE.dll

```
public static class AboutBOOSE
```

Inheritance

object ← AboutBOOSE

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

ARRAYLIMIT

```
public const int ARRAYLIMIT = 2
```

Field Value

int

BODYLIMIT

```
public const int BODYLIMIT = 5
```

Field Value

int

COMPOUNCOMMANDLIMIT

```
public const int COMPOUNCOMMANDLIMIT = 1
```

Field Value

int

MAXITTERATIONS

```
public const int MAXITTERATIONS = 200
```

Field Value

int

MAXPROGRAMSIZE

```
public const int MAXPROGRAMSIZE = 20
```

Field Value

int

METHODLIMIT

```
public const int METHODLIMIT = 1
```

Field Value

int

SIZELIMIT

```
public const int SIZELIMIT = 150
```

Field Value

int

VARIABLELIMIT

```
public const int VARIABLELIMIT = 5
```

Field Value

int

Properties

Version

Get version of BOOSE

```
public static float Version { get; }
```

Property Value

float

Methods

about()

```
public static string about()
```

Returns

string

Class Array

Namespace: BOOSE

Assembly: BOOSE.dll

An array of integers or reals. array int myArray 10,2\narray real prices 10\n[array][(type)][(size)][(optional dimentions, 1 if not given)]

```
public class Array : Evaluation, ICommand
```

Inheritance

object ← Command ← Evaluation ← Array

Implements

ICommand

Derived

Peek, Poke

Inherited Members

Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Array()

Contructor calls restrictions.

```
public Array()
```

Fields

PEEK

```
protected const bool PEEK = false
```

Field Value

bool

POKE

```
public const bool POKE = true
```

Field Value

bool

column

```
protected int column
```

Field Value

int

columnS

```
protected string columnS
```

Field Value

string

columns

```
protected int columns
```

Field Value

int

intArray

```
protected int[,] intArray
```

Field Value

int[,]

peekVar

```
protected string peekVar
```

Field Value

string

pokeValue

```
protected string pokeValue
```

Field Value

string

realArray

```
protected double[,] realArray
```

Field Value

double[,]

row

```
protected int row
```

Field Value

int

rowS

```
protected string rowS
```

Field Value

string

rows

```
protected int rows
```

Field Value

int

type

```
protected string type
```

Field Value

string

valueInt

```
protected int valueInt
```

Field Value

int

valueReal

```
protected double valueReal
```

Field Value

double

Properties

Columns

Dimensions of the array, 1 is default if no parameter is supplied.

```
protected int Columns { get; }
```

Property Value

int

Rows

Size of the array.

```
protected int Rows { get; }
```

Property Value

int

Methods

ArrayRestrictions()

```
public void ArrayRestrictions()
```

CheckParameters(string[])

Checks that there are 3 or 4 parameters.. array (size) (optional dimension)

```
public override void CheckParameters(string[] parameterList)
```

Parameters

`parameterList` string[]

Compile()

Ensure array is correctly declared

```
public override void Compile()
```

Exceptions

CommandException

Execute()

Evaluate any expression and store the evaluatedExpression result in instance data for sub classes. Expressions are evaluated at runtime by a StoredProgram object (Program)

```
public override void Execute()
```

GetIntArray(int, int)

```
public virtual int GetIntArray(int row, int col)
```

Parameters

`row` int

`col` int

Returns

int

GetRealArray(int, int)

```
public virtual double GetRealArray(int row, int col)
```

Parameters

`row` int

`col` int

Returns

double

ProcessArrayParametersCompile(bool)

Peek and Poke are very similar in syntax but not identical, so processing them has to be done slightly differently.

```
protected virtual void ProcessArrayParametersCompile(bool peekOrPoke)
```

Parameters

`peekOrPoke` bool

Exceptions

CommandException

ProcessArrayParametersExecute(bool)

Set up the object with parsed row and column (from any expressions). Used by Peek and Poke at runtime.

```
protected virtual void ProcessArrayParametersExecute(bool peekOrPoke)
```

Parameters

`peekOrPoke` bool

Exceptions

CommandException

SetIntArray(int, int, int)

```
public virtual void SetIntArray(int val, int row, int col)
```

Parameters

`val` int

`row` int

col int

SetRealArray(double, int, int)

```
public virtual void SetRealArray(double val, int row, int col)
```

Parameters

val double

row int

col int

Class BOOSEException

Namespace: BOOSE

Assembly: BOOSE.dll

Generic BOOSE Language exception Extends Exception

```
public class BOOSEException : Exception, ISerializable
```

Inheritance

object ← Exception ← BOOSEException

Implements

ISerializable

Derived

CanvasException, CommandException, FactoryException, ParserException, RestrictionException, StoredProgramException, VarException

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

BOOSEException(string)

```
public BOOSEException(string msg)
```

Parameters

`msg` string

Class Boolean

Namespace: BOOSE

Assembly: BOOSE.dll

Boolean datatype, also used for Conditional Commands.

```
public class Boolean : Evaluation, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean

Implements

ICommand

Derived

ConditionalCommand

Inherited Members

Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.CheckParameters(string[]) , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Boolean()

```
public Boolean()
```

Properties

BoolValue

```
public bool BoolValue { get; set; }
```

Property Value

bool

Methods

Compile()

```
public override void Compile()
```

Execute()

base.execute() will deliver a "true" or "flase" so convert that to an actual boolean value.

```
public override void Execute()
```

Exceptions

CommandException

Thrown if not a valid boolean expression.

Class Call

Namespace: BOOSE

Assembly: BOOSE.dll

Command to call a previously defined method. method int testMethod\n int one, int two\n testMethod = one* two\nend method\nint global = 55\ncall testMethod 10 15

```
public sealed class Call : CompoundCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← CompoundCommand ← Call

Implements

ICommand

Inherited Members

CompoundCommand.CorrespondingCommand , CompoundCommand.CheckParameters(string[]) , ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , Boolean.BoolValue , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Fields

methodName

```
protected string methodName
```

Field Value

string

Methods

Compile()

Override base.compile() because a variable always has variable = expression and this is just expression.

```
public override void Compile()
```

Exceptions

VarException

Execute()

Called when program is executed. Determines if condition is true or false. If false then jumps to correnspondingn "end" command.

```
public override void Execute()
```

Exceptions

CommandException

Class Canvas

Namespace: BOOSE

Assembly: BOOSE.dll

Abstract class that implements CanvasInterface. See CanvasInterface documentation for what to implement. This class handles all the drawing on your system according to above documentation. BOOSE does not by default specify a default drawing Canvas (only this).

```
public class Canvas : ICanvas
```

Inheritance

object ← Canvas

Implements

ICanvas

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Canvas()

```
public Canvas()
```

Fields

background_colour

```
protected Color background_colour
```

Field Value

Color

Properties

PenColour

Get/Set the Pencolour for next drawing operation using a native colour datatype. Cast to relevant type.

```
public virtual object PenColour { get; set; }
```

Property Value

object

Xpos

X position of next drawing operation.

```
public virtual int Xpos { get; set; }
```

Property Value

int

Ypos

Y position of next drawing position

```
public virtual int Ypos { get; set; }
```

Property Value

int

Methods

Circle(int, bool)

Draw a circle at cursor position of radius.

```
public virtual void Circle(int radius, bool filled)
```

Parameters

`radius` int

Radius of circle.

`filled` bool

If True circle is drawn filled, outline if false.

Clear()

Fill the background in the default colour.

```
public virtual void Clear()
```

DrawTo(int, int)

Draw a line using the current pen from the last drawingf position to the specified position and move the cursor position to the provided x,y

```
public virtual void DrawTo(int x, int y)
```

Parameters

`x` int

specified X position.

`y` int

specified Y position.

MoveTo(int, int)

Move the X and Y of the next drawing operation.

```
public virtual void MoveTo(int x, int y)
```

Parameters

`x` int

X position of cursor.

`y` int

Y position of cursor.

Rect(int, int, bool)

Draw a rectangle at cursor position of width and height.

```
public virtual void Rect(int width, int height, bool filled)
```

Parameters

`width` int

`height` int

`filled` bool

Rectangle(int, int, bool)

```
public virtual void Rectangle(int width, int height, bool filled)
```

Parameters

`width` int

`height` int

`filled` bool

Reset()

Reset drawing cursor to 0,0 and reset pen to default.

```
public virtual void Reset()
```

Set(int, int)

Set output display size. This method should create whatever drawing display you intend to use of the size specified.

```
public virtual void Set(int width, int height)
```

Parameters

`width` int

`height` int

SetColour(int, int, int)

Set the pen colour using rgb values.

```
public virtual void SetColour(int red, int green, int blue)
```

Parameters

`red` int

`green` int

`blue` int

Tri(int, int)

Draw a triangle in the bounding rectangle.

```
public virtual void Tri(int width, int height)
```

Parameters

`width` int

Width of bounding rectangle.

`height` int

Height of bounding rectangle.

WriteText(string)

Draws text on the output window at the cursor position

```
public virtual void WriteText(string text)
```

Parameters

`text` string

String to output

getBitmap()

Get the drawing Object of whatever native type. Returned a Object so it can be cast to native type. Use this to get native drawing type so that it can be displayed or output.

```
public virtual object getBitmap()
```

Returns

object

Class CanvasCommand

Namespace: BOOSE

Assembly: BOOSE.dll

Abstract class that is inherited by derived canvas commands that draw on a canvas. It stores the x and y positions of the last drawing operation.

```
public abstract class CanvasCommand : Command, ICommand
```

Inheritance

object ← Command ← CanvasCommand

Implements

ICommand

Derived

CommandOneParameter

Inherited Members

Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.CheckParameters(string[]) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.Execute() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CanvasCommand()

```
public CanvasCommand()
```

CanvasCommand(ICanvas)

Check the parsed parameters match what are expected, i.e. if a command requires two parameters has it got two parameters?

```
public CanvasCommand(ICanvas c)
```

Parameters

`c` ICanvas

Fields

canvas

```
protected ICanvas canvas
```

Field Value

ICanvas

xPos

```
protected int xPos
```

Field Value

int

yPos

```
protected int yPos
```

Field Value

int

Properties

Canvas

get/set Canvas object this command is associated with (useful

```
public ICanvas Canvas { get; set; }
```

Property Value

ICanvas

Class CanvasException

Namespace: BOOSE

Assembly: BOOSE.dll

Exception that should be thrown by operations that fail on the Canvas.

```
public class CanvasException : BOOSEException, ISerializable
```

Inheritance

object ← Exception ← BOOSEException ← CanvasException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CanvasException(string)

```
public CanvasException(string msg)
```

Parameters

`msg` string

See Also

BOOSEException

Class Cast

Namespace: BOOSE

Assembly: BOOSE.dll

Cast real values to int values. Syntax: cast *intVarrealVar* Both variables must be previously defined. real circ = 100.123\nint c\ncast c circ\ncircle c

```
public sealed class Cast : Command, ICommand
```

Inheritance

object ← Command ← Cast

Implements

ICommand

Inherited Members

Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Methods

CheckParameters(string[])

Derived commandas must provide a method to check that their parameters are ok and throw relevant exceptions if not.

```
public override void CheckParameters(string[] parameter)
```

Parameters

`parameter` string[]

Compile()

Called when Command added to the Program

```
public override void Compile()
```

Execute()

Generic Execute() checks a command's parameter list and converts any variables and expressions to literal values Should be called (base.Execute() from derived Command classes before the derived Command uses the parameters to do its job. Derived Command should check it has the correct number of parameters and throw a CommandException if not.

```
public override void Execute()
```

Class Circle

Namespace: BOOSE

Assembly: BOOSE.dll

Draw a circle round the current cursor position. circle 50Canvas exception if a non-integer is passed.

```
public sealed class Circle : CommandOneParameter, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← Circle

Implements

ICommand

Inherited Members

CanvasCommand.Canvas , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Circle()

blank constructor for factory instantiation.

```
public Circle()
```

Circle(Canvas, int)

draw from current position to x, y. cursor left at x,y

```
public Circle(Canvas c, int radius)
```

Parameters

`c` Canvas

`radius` int

Methods

CheckParameters(string[])

overridden generic one parameter message to say radius and not p1.

```
public override void CheckParameters(string[] parameterList)
```

Parameters

`parameterList` string[]

Exceptions

CommandException

Execute()

Execute the command

```
public override void Execute()
```

Class Command

Namespace: BOOSE

Assembly: BOOSE.dll

Abstract class for BOOSE commands. Can handle a parameter list or expression. This class will seperate the parameters or expression from the command and store in the "ParameterList". The Command's Compile() method is called by the parser and this is expected to set the Command's parameters. BOOSE implementation of commands makes definitions "sealed". Sealed classes cannot be inherited, so you can't use what's there to remove the restrictions, you will have to totally rewrite them.

```
public abstract class Command : ICommand
```

Inheritance

object ← Command

Implements

ICommand

Derived

CanvasCommand, Cast, Evaluation

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Command()

designed to be used with ProgramFactory so should not call constructor

```
public Command()
```

Fields

IsDouble

```
protected bool IsDouble
```

Field Value

bool

parameterList

```
protected string parameterList
```

Field Value

string

parameters

```
protected string[] parameters
```

Field Value

string[]

paramsint

```
protected int[] paramsint
```

Field Value

int[]

program

```
protected StoredProgram program
```

Field Value

StoredProgram

Properties

Name

returns the name type of this command as a string.

```
public string Name { get; }
```

Property Value

string

ParameterList

```
public string ParameterList { get; }
```

Property Value

string

Parameters

```
public string[] Parameters { get; set; }
```

Property Value

string[]

Paramsint

```
public int[] Paramsint { get; set; }
```

Property Value

int[]

Program

```
public StoredProgram Program { get; set; }
```

Property Value

StoredProgram

Methods

CheckParameters(string[])

Derived commandas must provide a method to check that their parameters are ok and throw relevant exceptions if not.

```
public abstract void CheckParameters(string[] parameter)
```

Parameters

`parameter` string[]

Compile()

Called when Command added to the Program

```
public virtual void Compile()
```

Execute()

Generic Execute() checks a command's parameter list and converts any variables and expressions to literal values Should be called (base.Execute() from derived Command classes before the derived Command uses the parameters to do its job. Derived Command should check it has the correct number of parameters and throw a CommandException if not.

```
public virtual void Execute()
```

ProcessParameters(string)

Converts a string containing parameters to seperate parameters. Given a raw, single string parameter list, seperated by commas this splits into seperate strings and returns in the out Params.

```
public int ProcessParameters(string ParameterList)
```

Parameters

`ParameterList` string

Returns

int

Number of parameters

Set(StoredProgram, string)

Set a Command Object

```
public virtual void Set(StoredProgram Program, string Params)
```

Parameters

`Program` StoredProgram

Reference to valid StoredProgram

`Params` string

Original parameter list e.g. "num1,num2"

ToString()

returns name and parameter list as a string suitable for reparsing.

```
public override string ToString()
```

Returns

string

Class CommandException

Namespace: BOOSE

Assembly: BOOSE.dll

Exception genereted by the StoredProgram class when a command fails

```
public class CommandException : BOOSEException, ISerializable
```

Inheritance

object ← Exception ← BOOSEException ← CommandException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CommandException(string)

```
public CommandException(string msg)
```

Parameters

`msg` string

Class CommandFactory

Namespace: BOOSE

Assembly: BOOSE.dll

```
public class CommandFactory : ICommandFactory
```

Inheritance

object ← CommandFactory

Implements

ICommandFactory

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

MakeCommand(string)

Make a Command object from the String passed.

To create additional commands create a Factory that extends this class and which create a MakeCommand() to make your new commands. Call base.MakeCommand() to make the above commands. Replace the entire Factory buy impelementing the IFactory interface.

```
public virtual ICommand MakeCommand(string commandType)
```

Parameters

`commandType` string

String holding command to be created. Case is unimportant and it is trimmed.

Returns

ICommand

ICommand object if successful

Exceptions

FactoryException

Thrown if no such command.

Class CommandOneParameter

Namespace: BOOSE

Assembly: BOOSE.dll

Commands with one parameter. The command system adds one parameter at a time. Can be used by any one parameter command.

```
public abstract class CommandOneParameter : CanvasCommand, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter

Implements

ICommand

Derived

Circle, CommandTwoParameters, Rect

Inherited Members

CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , CanvasCommand.Canvas , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.Execute() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CommandOneParameter()

```
public CommandOneParameter()
```

CommandOneParameter(Canvas)

Immediate execute move drawing cursor to x, y

```
public CommandOneParameter(Canvas c)
```

Parameters

c Canvas

Fields

param1

```
protected int param1
```

Field Value

int

param1unprocessed

```
protected string param1unprocessed
```

Field Value

string

Methods

CheckParameters(string[])

Attempt to get two integer parameters throw applicationException if not

```
public override void CheckParameters(string[] parameterList)
```

Parameters

`parameterList` string[]

Class CommandThreeParameters

Namespace: BOOSE

Assembly: BOOSE.dll

Commands with two parameters. Each command has an Xpos and Ypos that operates from the current cursor position.

```
public abstract class CommandThreeParameters : CommandTwoParameters, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← CommandTwoParameters ← CommandThreeParameters

Implements

ICommand

Derived

PenColour

Inherited Members

CommandTwoParameters.param2 , CommandTwoParameters.param2unprocessed , CommandOneParameter.param1 , CommandOneParameter.param1unprocessed , CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , CanvasCommand.Canvas , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.Execute() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CommandThreeParameters()

```
public CommandThreeParameters()
```

CommandThreeParameters(Canvas)

Immediate execute move drawing cursor to x, y

```
public CommandThreeParameters(Canvas c)
```

Parameters

c Canvas

Fields

param3

```
protected int param3
```

Field Value

int

param3unprocessed

```
protected string param3unprocessed
```

Field Value

string

Methods

CheckParameters(string[])

Attempt to get two integer parameters throw applicationException if not

```
public override void CheckParameters(string[] parameterList)
```

Parameters

`parameterList` string[]

Class CommandTwoParameters

Namespace: BOOSE

Assembly: BOOSE.dll

Commands with two parameters. Each command has an Xpos and Ypos that operates from the current cursor position.

```
public abstract class CommandTwoParameters : CommandOneParameter, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← CommandTwoParameters

Implements

ICommand

Derived

CommandThreeParameters, DrawTo, MoveTo

Inherited Members

CommandOneParameter.param1 , CommandOneParameter.param1unprocessed , CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , CanvasCommand.Canvas , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.Execute() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CommandTwoParameters()

```
public CommandTwoParameters()
```

CommandTwoParameters(Canvas)

Immediate execute move drawing cursor to x, y

```
public CommandTwoParameters(Canvas c)
```

Parameters

`c` Canvas

Fields

param2

```
protected int param2
```

Field Value

int

param2unprocessed

```
protected string param2unprocessed
```

Field Value

string

Methods

CheckParameters(string[])

Attempt to get two integer parameters throw applicationException if not

```
public override void CheckParameters(string[] parameterList)
```

Parameters

`parameterList` string[]

Class CompoundCommand

Namespace: BOOSE

Assembly: BOOSE.dll

CompoundCommand is a building block for shared functionality between CompoundCommands, such as IF, WHILE, FOR ,ELSE and ENDS Classes the inherit from here tend to be "sealed" so that they cnnot be inherited from to manipulate restrictions.

```
public class CompoundCommand : ConditionalCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← CompoundCommand

Implements

ICommand

Derived

Call, Else, End, If, Method, While

Inherited Members

ConditionalCommand.endLineNumber , ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , ConditionalCommand.Execute() , Boolean.BoolValue , Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CompoundCommand()

```
public CompoundCommand()
```

Properties

CorrespondingCommand

```
public ConditionalCommand CorrespondingCommand { get; set; }
```

Property Value

ConditionalCommand

Methods

CheckParameters(string[])

Does nothing in this case.

```
public override void CheckParameters(string[] parameter)
```

Parameters

`parameter` string[]

Compile()

Override base.compile() because a variable always has variable = expression and this is just expression.

```
public override void Compile()
```

Exceptions

VarException

ReduceRestrictions()

```
protected void ReduceRestrictions()
```

Class ConditionalCommand

Namespace: BOOSE

Assembly: BOOSE.dll

Used for Ifs, Whiles and Fors

```
public class ConditionalCommand : Boolean, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand

Implements

ICommand

Derived

CompoundCommand, For

Inherited Members

Boolean.BoolValue , Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.CheckParameters(string[]) , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Fields

endLineNumber

```
protected int endLineNumber
```

Field Value

int

Properties

CondType

Link end [type of conditional] to conditional, so compiler can tell if end end does not match

```
public ConditionalCommand.conditionalTypes CondType { get; set; }
```

Property Value

ConditionalCommand.conditionalTypes

Condition

Condition for If and While.

```
public bool Condition { get; set; }
```

Property Value

bool

EndLineNumber

Line number of corresponding "end" command for this conditional command (so when executed it can jump straight to it).

```
public int EndLineNumber { get; set; }
```

Property Value

int

LineNumber

Line number of this conditional command.

```
public int LineNumber { get; set; }
```

Property Value

int

ReturnLineNumber

```
public int ReturnLineNumber { get; set; }
```

Property Value

int

Methods

Compile()

Override base.compile() because a variable always has variable = expression and this is just expression.

```
public override void Compile()
```

Exceptions

VarException

Execute()

Called when program is executed. Determines if condition is true or false. If false then jumps to correnssponding "end" command.

```
public override void Execute()
```

Exceptions

CommandException

Enum ConditionalCommand.conditionalTypes

Namespace: BOOSE

Assembly: BOOSE.dll

Enumerated type to represent individual conditional commands.

```
public enum ConditionalCommand.conditionalTypes
```

Fields

`commFor = 2`

`commIF = 0`

`commWhile = 1`

Class DrawTo

Namespace: BOOSE

Assembly: BOOSE.dll

Draw a line from to current cursor position to the provided x,y position. drawto 200,200

```
public class DrawTo : CommandTwoParameters, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← CommandTwoParameters ← DrawTo

Implements

ICommand

Inherited Members

CommandTwoParameters.param2 , CommandTwoParameters.param2unprocessed , CommandTwoParameters.CheckParameters(string[]) , CommandOneParameter.param1 , CommandOneParameter.param1unprocessed , CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , CanvasCommand.Canvas , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

DrawTo()

blank constructor for factory instantiation.

```
public DrawTo()
```

DrawTo(Canvas, int, int)

draw from current position to x, y. cursor left at x,y

```
public DrawTo(Canvas c, int x, int y)
```

Parameters

c Canvas

x int

x position

y int

y position

Methods

Execute()

Execute the command

```
public override void Execute()
```

Class Else

Namespace: BOOSE

Assembly: BOOSE.dll

An Else is like an Endif but it pops the corresponding If off the stack and pushes itself on so that it becomes the corresponding command to the end

```
public class Else : CompoundCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← CompoundCommand ← Else

Implements

ICommand

Inherited Members

CompoundCommand.CorrespondingCommand , CompoundCommand.ReduceRestrictions() , ConditionalCommand.endLineNumber , ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , Boolean.BoolValue , Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Else()

```
public Else()
```

Properties

CorrespondingEnd

```
public End CorrespondingEnd { get; set; }
```

Property Value

End

Methods

CheckParameters(string[])

Does nothing in this case.

```
public override void CheckParameters(string[] parameter)
```

Parameters

`parameter` string[]

Compile()

Override base.compile() because a variable always has variable = expression and this is just expression.

```
public override void Compile()
```

Exceptions

VarException

Execute()

Called when program is executed. Determines if condition is true or false. If false then jumps to correnspondingen "end" command.

```
public override void Execute()
```

Exceptions

CommandException

Class End

Namespace: BOOSE

Assembly: BOOSE.dll

End command for If/While/For/Method. BOOSE has only one End type command which has a parameter to say what it belongs so eg. "end if". End checks Boose types for the ciorresponding command (If/While/For/Method) and will likely require a complete rewrite to replcae those commands. It would probbaly be easier to replace Parser, StoredProgram, CompoundCommand and this. Look at the metghods each provides, if you like, and try and fill in their fucntionality.

```
public sealed class End : CompoundCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← CompoundCommand ← End

Implements

ICommand

Inherited Members

CompoundCommand.CorrespondingCommand , CompoundCommand.CheckParameters(string[]) , ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , Boolean.BoolValue , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Constructors

End()

```
public End()
```

Methods

Compile()

Checks that this end matches the last condition (i.e. if this is an end if then the command it pops off the stack is an "if".

```
public override void Compile()
```

Exceptions

CommandException

Thrown if the corresponding command does not match.

Execute()

Ifs are simple, do nothing. Whiles jumps back to the corresponding while command and For gets the from, to and step, checks they will result in a loop and determines whether the count has run out. If it hasn't it jumps to the line after the for.

```
public override void Execute()
```

Exceptions

CommandException

Thrown if invalid for loop.

Class Evaluation

Namespace: BOOSE

Assembly: BOOSE.dll

An evaluation encompasses all things that can have a value, such as variables, expressions and conditions used in loops and ifs.

```
public class Evaluation : Command, ICommand
```

Inheritance

object ← Command ← Evaluation

Implements

ICommand

Derived

Array, Boolean, Int, Real, Write

Inherited Members

Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Evaluation()

Blank constructor,

```
public Evaluation()
```

Fields

evaluatedExpression

```
protected string evaluatedExpression
```

Field Value

string

expression

```
protected string expression
```

Field Value

string

value

```
protected int value
```

Field Value

int

varName

```
protected string varName
```

Field Value

string

Properties

Expression

The right hand side of an expression in a variable initialisetion or variable change of value, or a boolean in a conditional command

```
public string Expression { get; set; }
```

Property Value

string

Local

Is this a local variable/method variable? This is so that variables of the same name can exist in defferent scopes.

```
public bool Local { get; set; }
```

Property Value

bool

Value

Value of variable once the expression has been calculated. Virtual so that variables of other types can override it.

```
public virtual int Value { get; set; }
```

Property Value

int

VarName

Name of variable as a string (as opposed to Name which is the name of the class, such as Int, Real or Boolean).

```
public string VarName { get; set; }
```

Property Value

string

Methods

CheckParameters(string[])

Does nothing in this case.

```
public override void CheckParameters(string[] parameterList)
```

Parameters

`parameterList` string[]

Compile()

```
public override void Compile()
```

Execute()

Evaluate any expression and store the evaluatedExpression result in instance data for sub classes. Expressions are evaluated at runtime by a StoredProgram object (Program)

```
public override void Execute()
```

ProcessExpression(string)

Takes a complete expression with left hand side and right hand side and returns only the right hand side (i.e. the bit after the =).

```
public virtual string ProcessExpression(string Expression)
```

Parameters

`Expression` string

Expression to evaluate

Returns

string

Right hand side of expression (the bit after the =)

Exceptions

VarException

Var exception if it isn't an expression

Class FactoryException

Namespace: BOOSE

Assembly: BOOSE.dll

exception genereted by the StoredProgram class

```
public class FactoryException : BOOSEException, ISerializable
```

Inheritance

object ← Exception ← BOOSEException ← FactoryException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

FactoryException(string)

```
public FactoryException(string msg)
```

Parameters

`msg` string

Class For

Namespace: BOOSE

Assembly: BOOSE.dll

For command. Adds processing at compile() and execute() to process a for command.

```
public sealed class For : ConditionalCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← For

Implements

ICommand

Inherited Members

ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , Boolean.BoolValue , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.CheckParameters(string[]) , Evaluation.ProcessExpression(string) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Constructors

For()

```
public For()
```

Properties

From

Where counting starts.

```
public int From { get; set; }
```

Property Value

int

LoopControlV

For loop uses a variable to do its counting.

```
public Evaluation LoopControlV { get; }
```

Property Value

Evaluation

Step

What do do each count. If none is specified it is set to +1t.

```
public int Step { get; set; }
```

Property Value

int

To

Where counting ends.

```
public int To { get; set; }
```

Property Value

int

Methods

Compile()

Takes the standard parameter list and extracts the LCV name, "from","to" and "step". If the LCV does not exists then it creates it. Otherwise it gets it from the variable table.

```
public override void Compile()
```

Execute()

Evaluate the from, to and step (as they could be variables). Update the LCV value.

```
public override void Execute()
```

Exceptions

StoredProgramException

Thrown if problem with expression.

Interface ICanvas

Namespace: BOOSE

Assembly: BOOSE.dll

Implement ICanvas for your BOOSE renderer. It has an Xpos and Ypos of the current cursor position, and a pen colour. Your class should implement the methods below to draw on its "bitmap" (i.e. it may not be a bitmap, it could draw in ASCII text for example).

```
public interface ICanvas
```

Properties

PenColour

Get/Set the Pencolour for next drawing operation using a native colour datatype. Cast to relevant type.

```
object PenColour { get; set; }
```

Property Value

object

Xpos

X position of next drawing operation.

```
int Xpos { get; set; }
```

Property Value

int

Ypos

Y position of next drawing position

```
int Ypos { get; set; }
```

Property Value

int

Methods

Circle(int, bool)

Draw a circle at cursor position of radius.

```
void Circle(int radius, bool filled)
```

Parameters

`radius` int

Radius of circle.

`filled` bool

If True circle is drawn filled, outline if false.

Clear()

Fill the background in the default colour.

```
void Clear()
```

DrawTo(int, int)

Draw a line using the current pen from the last drawingf position to the specified position and move the cursor position to the provided x,y

```
void DrawTo(int x, int y)
```

Parameters

`x` int

specified X position.

`y` int

specified Y position.

MoveTo(int, int)

Move the X and Y of the next drawing operation.

```
void MoveTo(int x, int y)
```

Parameters

`x` int

X position of cursor.

`y` int

Y position of cursor.

Rect(int, int, bool)

Draw a rectangle at cursor position of width and height.

```
void Rect(int width, int height, bool filled)
```

Parameters

`width` int

`height` int

`filled` bool

Reset()

Reset drawing cursor to 0,0 and reset pen to default.

```
void Reset()
```

Set(int, int)

Set output display size. This method should create whatever drawing display you intend to use of the size specified.

```
void Set(int width, int height)
```

Parameters

`width` int

`height` int

SetColour(int, int, int)

Set the pen colour using rgb values.

```
void SetColour(int red, int green, int blue)
```

Parameters

`red` int

`green` int

`blue` int

Tri(int, int)

Draw a triangle in the bounding rectangle.

```
void Tri(int width, int height)
```

Parameters

`width` int

Width of bounding rectangle.

`height` int

Height of bounding rectangle.

WriteText(string)

Draws text on the output window at the cursor position

```
void WriteText(string text)
```

Parameters

`text` string

String to output

getBitmap()

Get the drawing Object of whatever native type. Returned a Object so it can be cast to native type. Use this to get native drawing type so that it can be displayed or output.

```
object getBitmap()
```

Returns

object

Interface ICommand

Namespace: BOOSE

Assembly: BOOSE.dll

Interface for new commands. Any new Command class should implement this interface and be generated by a CommandFactory that implements the ICommandFactory interface. Contains methods to set the Command up after its creation, manage its parameters when it is compiled and execute it when it is called.

```
public interface ICommand
```

Methods

CheckParameters(string[])

Checks that a Command has the right number of parameters and throws a CommandException if not.

```
void CheckParameters(string[] Parameters)
```

Parameters

`Parameters` string[]

Compile()

Called before the command is run.

```
void Compile()
```

Execute()

Called to run the command.

```
void Execute()
```

Set(StoredProgram, string)

Set a Command Object

```
void Set(StoredProgram Program, string Params)
```

Parameters

`Program` StoredProgram

Reference to valid StoredProgram

`Params` string

Original parameter list e.g. "num1,num2"

Interface ICommandFactory

Namespace: BOOSE

Assembly: BOOSE.dll

To add commands to BOOSE you must create a CommandFactory that uses this interface. You should extend the existing BOOSE:CommandFactory (which implements this interface) and then implement the MakeCommand() method. It shoudld create a new command object based on the string passed. Any standard BOOSE commands can then be made by calling base.MakeCommand();

```
public interface ICommandFactory
```

Methods

MakeCommand(string)

Make a BOOSE Command based on the string passed to it.

```
ICommand MakeCommand(string commandType)
```

Parameters

`commandType` string

Returns

ICommand

Reference to new Command object.

Interface IEvaluation

Namespace: BOOSE

Assembly: BOOSE.dll

IEvaluation adds properties to get and set an Evaluations name, value and expression. An evaluation can be a variable declaration, such as "int total" or "real area". It can be change of value of a variable, such as "total = total + 1", the part after the "=" is an referred to as an "expression". In the first two cases the parser will generate an Int object and a Real Object. In the third case it will generate a "Evaluation" object.

```
public interface IEvaluation : ICommand
```

Inherited Members
ICommand.Set(StoredProgram, string) , ICommand.Compile() , ICommand.Execute() , ICommand.CheckParameters(string[])

Properties

Expression

```
string Expression { get; set; }
```

Property Value

string

Value

```
object Value { get; set; }
```

Property Value

object

VarName

```
string VarName { get; set; }
```

Property Value

string

Interface IParser

Namespace: BOOSE

Assembly: BOOSE.dll

The Parser Class takes a BOOSE program as a String with each command seperated by '\n' and creates command objects for each valid command and stores them in the passed in StoredProgram. Exceptions are generated for any syntax errors. When each valid command is generated its Compile() method is called. The valid command will have its parameters processed and any variables identified. It is the role of StoredProgram to run the commands.

```
public interface IParser
```

Methods

ParseCommand(string)

Parse a single command. Takes a single line of a BOOSE program "-command- -parameterlist-" or "-variable- = -expression-" It seperates the command from the parameter list or the variable from the expression. An Object of the command is made. its parameter list/expression is set and its Compile() method is called.

```
ICommand ParseCommand(string Line)
```

Parameters

`Line` string

"-command- -parameterlist-" or "-variable- = -expression-"

Returns

ICommand

ICommand Object which can then be executed

ParseProgram(string)

Parse the entrire program, adding valid command objects to the StoredProgram. An errorlist string is generated containing any systax error messages.

```
void ParseProgram(string program)
```

Parameters

`program` string

big string of a complete program seperated by newlines

Interface IStoredProgram

Namespace: BOOSE

Assembly: BOOSE.dll

```
public interface IStoredProgram
```

Properties

PC

property for int pc the program counter

```
int PC { get; set; }
```

Property Value

int

Methods

Add(Command)

adds Command to program, calls Command.compile() Any new command needs to implement a compile() method which will be called here when the command is compiled and added.

```
int Add(Command C)
```

Parameters

`C` Command

Returns

int

index at which member was added

AddVariable(Evaluation)

Add a Var object to the StoredProgram.

```
void AddVariable(Evaluation Variable)
```

Parameters

`Variable` Evaluation

Commandsleft()

Are there any commands left to execute in the program? i.e. pc (Program Counter) has not yet reached the end of the program

```
bool Commandsleft()
```

Returns

bool

true if commands left to execute, false if the end has been reached

EvaluateExpression(string)

Evaluate the given expression by finding the values of any variables and passing the result as a String

```
string EvaluateExpression(string Exp)
```

Parameters

`Exp` string

Returns

string

Exceptions

StoredProgramException

Throws StoredProgramException is it cannot be evaluated. Use IsExpression() before calling to prevent this exception being thrown.

GetVarValue(string)

Return the String value of a variable. It should throw a StoredProgramException if attampt is made to retrieve a non-existant variable. i.e. don't try, check first.

```
string GetVarValue(string varName)
```

Parameters

`varName` string

Returns

string

IsExpression(string)

Determine if the passed in string is an evaluatable expression.

```
bool IsExpression(string expression)
```

Parameters

`expression` string

Expression to be tested.

Returns

bool

true is it is an expression.

ResetProgram()

Once a program has finished executing it needs to be reset (Program Counter set to zero)

```
void ResetProgram()
```

Run()

Attempt to execute the program, throws a StoredProgram if it cannot run. The parser object should have generated a runnable program before running.

```
void Run()
```

Exceptions

StoredProgramException

VariableExists(string)

Returns true if variable has been defined in this program, false if not.

```
bool VariableExists(string varName)
```

Parameters

`varName` string

Returns

bool

True if variable exists.

Class If

Namespace: BOOSE

Assembly: BOOSE.dll

If command. Adds processing at compile() and execute() to process an if command.

```
public sealed class If : CompoundCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← CompoundCommand ← If

Implements

ICommand

Inherited Members

CompoundCommand.CorrespondingCommand , CompoundCommand.CheckParameters(string[]) , CompoundCommand.Compile() , ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , ConditionalCommand.Execute() , Boolean.BoolValue , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Constructors

If()

```
public If()
```

Class Int

Namespace: BOOSE

Assembly: BOOSE.dll

Command Class to define and store integer values.

```
public sealed class Int : Evaluation, ICommand
```

Inheritance

object ← Command ← Evaluation ← Int

Implements

ICommand

Inherited Members

Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.CheckParameters(string[]) , Evaluation.ProcessExpression(string) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Int()

Contructor

```
public Int()
```

Methods

Compile()

If this is a variable declartation set its value and add it to the variable table in the StoredProgram. If it is an expression then nothing happens as it already exists in the variable table (that is handled within

StoredProgram).

```
public override void Compile()
```

Execute()

Base calculates a string evaluated expression. Now see if it's an integer value. Determines if a cast is needed for a real value.

```
public override void Execute()
```

Exceptions

StoredProgramException

Throws an exception is it can't parse an integer or if there is an attempt to parse a real value which requires a boose cast to force it.

Class Method

Namespace: BOOSE

Assembly: BOOSE.dll

Method declaration command. Overcoming method restrictions will require a complete rewrite. It would be simpler to do this after Parser and StoredProgram. You are getting to the stage where you need to do a big rewrite.

```
public class Method : CompoundCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← CompoundCommand ← Method

Implements

ICommand

Inherited Members

CompoundCommand.CorrespondingCommand , CompoundCommand.ReduceRestrictions() , ConditionalCommand.endLineNumber , ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , Boolean.BoolValue , Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Method()

The Call inserts it's line number in here so the method end can return.

```
public Method()
```

Properties

LocalVariables

Get parameter list of the method, treated as local variable. They will be deleted when execution leave the method.

```
public string[] LocalVariables { get; }
```

Property Value

string[]

MethodName

Get method name, so a parser can append it to local variables/parameters.

```
public string MethodName { get; }
```

Property Value

string

Type

Return type of method

```
public string Type { get; }
```

Property Value

string

Methods

CheckParameters(string[])

Does nothing in this case.

```
public override void CheckParameters(string[] parameter)
```

Parameters

`parameter` string[]

Compile()

Override base.compile() because a variable always has variable = expression and this is just expression.

```
public override void Compile()
```

Exceptions

VarException

Execute()

Called when program is executed. Determines if condition is true or false. If false then jumps to correnspondingn "end" command.

```
public override void Execute()
```

Exceptions

CommandException

Class MoveTo

Namespace: BOOSE

Assembly: BOOSE.dll

Set the cursor position, for subsequent drawing operations, to the provide x,y position.

```
public class MoveTo : CommandTwoParameters, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← CommandTwoParameters ← MoveTo

Implements

ICommand

Inherited Members

CommandTwoParameters.param2 , CommandTwoParameters.param2unprocessed , CommandTwoParameters.CheckParameters(string[]) , CommandOneParameter.param1 , CommandOneParameter.param1unprocessed , CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , CanvasCommand.Canvas , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

MoveTo()

Blank constructor for factory instantiation.

```
public MoveTo()
```

Methods

Execute()

Execute the moveto command, if sucessful the drawing cursor will be moved to the passed in x,y position.

```
public override void Execute()
```

Class Parser

Namespace: BOOSE

Assembly: BOOSE.dll

The parser takes a program, who's lines are seperated by "\n", and attempts to create a StoredProgram. Once the SToredProgram has been created it then can be run. As much processing that can be done is done here because it has no run penalty (other than starting the program). Command objects are created and their parameters processed as much as possible. Variables are checked for existance but expressions cannot be processed until runtime.

```
public class Parser : IParser
```

Inheritance

object ← Parser

Implements

IParser

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Parser(CommandFactory, StoredProgram)

Create Parser object with associated StoredProgram.

```
public Parser(CommandFactory Factory, StoredProgram Program)
```

Parameters

`Factory` CommandFactory

CommandFactory that the paser will call to make command objects..

`Program` StoredProgram

StoredProgram to add generated commands to.

Exceptions

ParserException

Thrown on syntax errors. message contains a complete list with line numbers, seperated by return characters.

Methods

ParseCommand(string)

Take a line and attempt to parse a BOOSE command. The command is split from its parameters. It determines if variables are being defined or updated. The Command Factory is called to make the Command object and its Compile() method is called to further process its parameters. Normal commands are of the form [command][space][parameter list] Variable declarations\redefinitions are of the form [variablename][=][expression] where expression is optional. To create your own parser it would be possible to just implement the IParser interface. You could extend Parser as well.

```
public virtual ICommand ParseCommand(string Line)
```

Parameters

`Line` string

Returns

ICommand

Exceptions

ParserException

Throws exceptions if an undefined variable is used in an expression.

ParseProgram(string)

The whole program is processed line by line. ParseCommand() is called for each line.

```
public virtual void ParseProgram(string program)
```

Parameters

`program` string

Exceptions

ParserException

Complete list of syntax errors with line numbers.

Class ParserException

Namespace: BOOSE

Assembly: BOOSE.dll

exception genereted by the Parser class to denote syntax errors

```
public class ParserException : BOOSEException, ISerializable
```

Inheritance

object ← Exception ← BOOSEException ← ParserException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

ParserException(string)

```
public ParserException(string msg)
```

Parameters

`msg` string

Class Peek

Namespace: BOOSE

Assembly: BOOSE.dll

Peek an array, i.e. get the value of an array cell. [variable] = peek [array name] [row] [optional column]

```
public class Peek : Array, ICommand
```

Inheritance

object ← Command ← Evaluation ← Array ← Peek

Implements

ICommand

Inherited Members

Array.PEEK , Array.POKE , Array.type , Array.rows , Array.columns , Array.valueInt , Array.valueReal , Array.intArray , Array.realArray , Array.pokeValue , Array.peekVar , Array.rowS , Array.columnS , Array.row , Array.column , Array.Rows , Array.Columns , Array.ArrayRestrictions() , Array.ProcessArrayParametersCompile(bool) , Array.ProcessArrayParametersExecute(bool) , Array.SetIntArray(int, int, int) , Array.SetRealArray(double, int, int) , Array.GetIntArray(int, int) , Array.GetRealArray(int, int) , Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Peek()

```
public Peek()
```

Methods

CheckParameters(string[])

Checks that there are 3 or 4 parameters.. array (size) (optional dimension)

```
public override void CheckParameters(string[] Parameters)
```

Parameters

`Parameters` string[]

Compile()

Ensure array is correctly declared

```
public override void Compile()
```

Exceptions

CommandException

Execute()

Evaluate any expression and store the evaluatedExpression result in instance data for sub classes.
Expressions are evaluated at runtime by a StoredProgram object (Program)

```
public override void Execute()
```

Class PenColour

Namespace: BOOSE

Assembly: BOOSE.dll

Set the pen for the next drawing operations to the provided r,g,b colour.

```
public class PenColour : CommandThreeParameters, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← CommandTwoParameters ← CommandThreeParameters ← PenColour

Implements

ICommand

Inherited Members

CommandThreeParameters.param3 , CommandThreeParameters.param3unprocessed , CommandThreeParameters.CheckParameters(string[]) , CommandTwoParameters.param2 , CommandTwoParameters.param2unprocessed , CommandOneParameter.param1 , CommandOneParameter.param1unprocessed , CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , CanvasCommand.Canvas , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Methods

Execute()

Generic Execute() checks a command's parameter list and converts any variables and expressions to literal values Should be called (base.Execute() from derived Command classes before the derived Command uses the parameters to do its job. Derived Command should check it has the correct number of parameters and throw a CommandException if not.

```
public override void Execute()
```

Class Poke

Namespace: BOOSE

Assembly: BOOSE.dll

Poke an array, i.e. set the value of an array cell. poke [array name] [row] [optional column] = [value]

```
public class Poke : Array, ICommand
```

Inheritance

object ← Command ← Evaluation ← Array ← Poke

Implements

ICommand

Inherited Members

Array.PEEK , Array.POKE , Array.type , Array.rows , Array.columns , Array.valueInt , Array.valueReal , Array.intArray , Array.realArray , Array.pokeValue , Array.peekVar , Array.rowS , Array.columnS , Array.row , Array.column , Array.Rows , Array.Columns , Array.ArrayRestrictions() , Array.ProcessArrayParametersCompile(bool) , Array.ProcessArrayParametersExecute(bool) , Array.SetIntArray(int, int, int) , Array.SetRealArray(double, int, int) , Array.GetIntArray(int, int) , Array.GetRealArray(int, int) , Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Poke()

```
public Poke()
```

Methods

CheckParameters(string[])

Checks that there are 3 or 4 parameters.. array (size) (optional dimension)

```
public override void CheckParameters(string[] parameter)
```

Parameters

`parameter` string[]

Compile()

Ensure array is correctly declared

```
public override void Compile()
```

Exceptions

CommandException

Execute()

Evaluate any expression and store the evaluatedExpression result in instance data for sub classes. Expressions are evaluated at runtime by a StoredProgram object (Program)

```
public override void Execute()
```

Set(StoredProgram, string)

Set a Command Object

```
public override void Set(StoredProgram Program, string Params)
```

Parameters

`Program` StoredProgram

Reference to valid StoredProgram

`Params` string

Original parameter list e.g. "num1,num2"

Class Real

Namespace: BOOSE

Assembly: BOOSE.dll

Class for a Real Number Variable.

```
public sealed class Real : Evaluation, ICommand
```

Inheritance

object ← Command ← Evaluation ← Real

Implements

ICommand

Inherited Members

Evaluation.Expression , Evaluation.VarName , Evaluation.Local , Evaluation.CheckParameters(string[]) , Evaluation.ProcessExpression(string) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Constructors

Real()

Contructor

```
public Real()
```

Properties

Value

Get double value for a real number.

```
public double Value { get; set; }
```

Property Value

double

Methods

Compile()

If this is a variable declartation set its value and add it to the variable table. If it is an expression then nothis happens as it already exists in the variable table.

```
public override void Compile()
```

Execute()

Base calculates a string evaluated expression. Now see if it's a double/real value.

```
public override void Execute()
```

Exceptions

StoredProgramException

Class Rect

Namespace: BOOSE

Assembly: BOOSE.dll

Draw a rectangle of provided with and heigh with the current cursor position being the top left corner.

```
public class Rect : CommandOneParameter, ICommand
```

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← Rect

Implements

ICommand

Inherited Members

CommandOneParameter.param1 , CommandOneParameter.param1unprocessed , CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , CanvasCommand.Canvas , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.Compile() , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Rect()

blank constructor for factory instantiation.

```
public Rect()
```

Rect(Canvas, int, int)

draw from current position to x, y. cursor left at x,y

```
public Rect(Canvas c, int width, int height)
```

Parameters

`c` Canvas

`width` int

`height` int

Methods

CheckParameters(string[])

overridden generic one parameter message to say radius and not p1.

```
public override void CheckParameters(string[] parameterList)
```

Parameters

`parameterList` string[]

Exceptions

CommandException

Execute()

Execute the command

```
public override void Execute()
```

Class RestrictionException

Namespace: BOOSE

Assembly: BOOSE.dll

Exception thrown when restrictions are breached. If you get this you need to write your own code for the facility.

```
public class RestrictionException : BOOSEException, ISerializable
```

Inheritance

object ← Exception ← BOOSEException ← RestrictionException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

RestrictionException(string)

```
public RestrictionException(string msg)
```

Parameters

`msg` string

Class StoredProgram

Namespace: BOOSE

Assembly: BOOSE.dll

A collection class for storing a program of Command objects, extends ArrayList to add a program counter and a flag to indicate that the syntax is ok and the program valid. Adds methods below to process variables and methods and implements flow of control for ifs and loops.

```
public class StoredProgram : ArrayList, IList, ICollection, IEnumerable,
ICloneable, IStoredProgram
```

Inheritance

object ← ArrayList ← StoredProgram

Implements

IList, ICollection, IEnumerable, ICloneable, IStoredProgram

Inherited Members

ArrayList.Adapter(IList) , ArrayList.Add(object) , ArrayList.AddRange(ICollection) , ArrayList.BinarySearch(int, int, object, IComparer) , ArrayList.BinarySearch(object) , ArrayList.BinarySearch(object, IComparer) , ArrayList.Clear() , ArrayList.Clone() , ArrayList.Contains(object) , ArrayList.CopyTo(Array) , ArrayList.CopyTo(Array, int) , ArrayList.CopyTo(int, Array, int, int) , ArrayList.FixedSize(ArrayList) , ArrayList.FixedSize(IList) , ArrayList.GetEnumerator() , ArrayList.GetEnumerator(int, int) , ArrayList.GetRange(int, int) , ArrayList.IndexOf(object) , ArrayList.IndexOf(object, int) , ArrayList.IndexOf(object, int, int) , ArrayList.Insert(int, object) , ArrayList.InsertRange(int, ICollection) , ArrayList.LastIndexOf(object) , ArrayList.LastIndexOf(object, int) , ArrayList.LastIndexOf(object, int, int) , ArrayList.ReadOnly(ArrayList) , ArrayList.ReadOnly(IList) , ArrayList.Remove(object) , ArrayList.RemoveAt(int) , ArrayList.RemoveRange(int, int) , ArrayList.Repeat(object, int) , ArrayList.Reverse() , ArrayList.Reverse(int, int) , ArrayList.SetRange(int, ICollection) , ArrayList.Sort() , ArrayList.Sort(IComparer) , ArrayList.Sort(int, int, IComparer) , ArrayList.Synchronized(ArrayList) , ArrayList.Synchronized(IList) , ArrayList.ToArray() , ArrayList.ToArray(Type) , ArrayList.TrimToSize() , ArrayList.Capacity , ArrayList.Count , ArrayList.IsFixedSize , ArrayList.IsReadOnly , ArrayList.IsSynchronized , ArrayList.this[int] , ArrayList.SyncRoot , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

StoredProgram(ICanvas)

Create a blank program which will output to the provided Canvas object. There are a lot of methods in here and it implements the IStoredProgram interface and extends ArrayList. This makes it easier to extend StoredProgram and reimplement the methods you need.

```
public StoredProgram(ICanvas canvas)
```

Parameters

`canvas` ICanvas

Properties

PC

Program Counter, points at the next command object.

```
public virtual int PC { get; set; }
```

Property Value

int

Methods

Add(Command)

Adds Command to a stored program

```
public virtual int Add(Command C)
```

Parameters

`C` Command

Returns

int

index at which member was added

AddMethod(Method)

Add a Method object to the methods table so it can be linked to a call.

```
public virtual void AddMethod(Method M)
```

Parameters

`M` Method

Method Object to add

AddVariable(Evaluation)

Add a variable/evaluation to the Stored Program.

```
public virtual void AddVariable(Evaluation Variable)
```

Parameters

`Variable` Evaluation

Evaluation Object

Commandsleft()

Are there any commands left to execute in the program? i.e. pc (Program Counter) has not yet reached the end of the program

```
public virtual bool Commandsleft()
```

Returns

bool

true if commands left to execute, false if the end has been reached

DeleteVariable(string)

If the passed in variable name exists then delete it.

```
public virtual void DeleteVariable(string varName)
```

Parameters

`varName` string

Variable name to delete.

EvaluateExpression(string)

Evaluate the given expression by finding the values of any variables and passing the final evaluated result as a String. Uses the DataTable Class Compute() method which can parse an expression as a string, but it doesn't know about boose variable so they have to be converted to values and the expression assembled with them.

```
public virtual string EvaluateExpression(string Exp)
```

Parameters

`Exp` string

Returns

string

Exceptions

StoredProgramException

Throws StoredProgramException is it cannot be evaluated. Use IsExpression() before calling to prevent this exception being thrown.

EvaluateExpressionWithString(string)

Evaluate an expression with a string. i.e. area + " cm^2"

```
public virtual string EvaluateExpressionWithString(string expression)
```

Parameters

`expression` string

Expression as a String

Returns

string

Resulting string to display.

FindVariable(Evaluation)

Finds the position of a Variable Object in the Variable table.

```
public virtual int FindVariable(Evaluation Variable)
```

Parameters

`Variable` Evaluation

Evaluation Object to find.

Returns

int

Position or -1 if not found

FindVariable(string)

Get a variable's position in the Variable table. The variable is expressed by its string name.

```
public virtual int FindVariable(string varName)
```

Parameters

`varName` string

The string name of the variable.

Returns

int

-1 if not found, otherwise the position.

GetMethod(string)

Get the method object from the StoredProgram's method table from its string name.

```
public virtual Method GetMethod(string MethodName)
```

Parameters

`MethodName` string

String name of method.

Returns

Method

Exceptions

StoredProgramException

Thrown if method does not exist

GetVarValue(string)

Returns a variables value from it string name.

```
public virtual string GetVarValue(string varName)
```

Parameters

`varName` string

Variable name.

Returns

string

String value of variable.

Exceptions

StoredProgramException

If variable is not found.

GetVariable(int)

Return the Variable object at the given index in the Variable table.

```
public virtual Evaluation GetVariable(int index)
```

Parameters

`index` int

Position in Variables table to extract.

Returns

Evaluation

Position of object in Variables Table.

Exceptions

StoredProgramException

Thrown if invalid index is passed.

GetVariable(string)

Get a Variable Object. Evaluation is the base class, it could itself be a subclass of the actual types (int, real boolean etc).

```
public virtual Evaluation GetVariable(string VarName)
```

Parameters

`VarName` string

Returns

Evaluation

Evaluation Object of the found type

Exceptions

StoredProgramException

Throws exception if cannot be found, should check that it is there first.

IsExpression(string)

```
public virtual bool IsExpression(string expression)
```

Parameters

`expression` string

Returns

bool

IsValidProgram()

Returns true if the StoredProgram contains a program with no syntaxErrors. Probably shouldn't try and tun one that does.

```
public bool IsValidProgram()
```

Returns

bool

NextCommand()

```
public virtual object NextCommand()
```

Returns

object

Pop()

Pop a compound command onto the stack. Used for whiles/if/fors/methods to move the PC.

```
public virtual ConditionalCommand Pop()
```

Returns

ConditionalCommand

Exceptions

StoredProgramException

Throws StoredProgramException if it can't pop, which is assumed to be because there is a compound statement without and end.

Push(ConditionalCommand)

Push a compound command onto the stack. Used for whiles/if/fors/methods to move the PC.

```
public virtual void Push(ConditionalCommand Com)
```

Parameters

`Com` ConditionalCommand

Compound Command Object

ResetProgram()

Once a program has finished executing it needs to be reset (Program Counter set to zero)

```
public virtual void ResetProgram()
```

Run()

Attempt to execute the program, throws a StoredProgram if it cannot run to completion. It is probably better to not try and run a program that passed an errorlist back from Parser.ParseProgram() but it will try. The parser object should have generated a runnable program before running. Attempts to detect infinite loops by detecting it has gione fround the run loop a lot.

```
public virtual void Run()
```

Exceptions

StoredProgramException

SetSyntaxStatus(bool)

Set the syntax status of this program, true = no errors, false = errors. Called by a parser that created the program.

```
public void SetSyntaxStatus(bool status)
```

Parameters

`status` bool

UpdateVariable(string, bool)

Update integer variable value.

```
public virtual void UpdateVariable(string varName, bool value)
```

Parameters

`varName` string

String name of variable.

`value` bool

New integer value.

UpdateVariable(string, double)

Update real variable value.

```
public virtual void UpdateVariable(string varName, double value)
```

Parameters

`varName` string

String name of variable.

`value` double

New real value.

UpdateVariable(string, int)

Update an existing variable's integer value if it is found, do nothing if not found.

```
public virtual void UpdateVariable(string varName, int value)
```

Parameters

`varName` string

String name of variable.

`value` int

Integer value to change the variable to.

VariableExists(string)

Returns true if the variable specified by its name exists.

```
public virtual bool VariableExists(string varName)
```

Parameters

`varName` string

String variable name.

Returns

bool

true if variable has been declared previously.

Class StoredProgramException

Namespace: BOOSE

Assembly: BOOSE.dll

exception genereted by the StoredProgram class

```
public class StoredProgramException : BOOSEException, ISerializable
```

Inheritance

object ← Exception ← BOOSEException ← StoredProgramException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

StoredProgramException(string)

```
public StoredProgramException(string msg)
```

Parameters

`msg` string

Class VarException

Namespace: BOOSE

Assembly: BOOSE.dll

Exception raised by variable commands.

```
public class VarException : BOOSEException, ISerializable
```

Inheritance

object ← Exception ← BOOSEException ← VarException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetObjectData(SerializationInfo, StreamingContext) , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

VarException(string)

```
public VarException(string msg)
```

Parameters

`msg` string

Class While

Namespace: BOOSE

Assembly: BOOSE.dll

While command. So far blank as it doesn't do anything beyond ConditionalCommand but it makes the code clearer (in the factory).

```
public sealed class While : CompoundCommand, ICommand
```

Inheritance

object ← Command ← Evaluation ← Boolean ← ConditionalCommand ← CompoundCommand ← While

Implements

ICommand

Inherited Members

CompoundCommand.CorrespondingCommand , CompoundCommand.CheckParameters(string[]) , CompoundCommand.Compile() , ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.ReturnLineNumber , ConditionalCommand.Execute() , Boolean.BoolValue , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.ProcessExpression(string) , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Constructors

While()

```
public While()
```

Class Write

Namespace: BOOSE

Assembly: BOOSE.dll

```
public class Write : Evaluation, ICommand
```

Inheritance

object ← Command ← Evaluation ← Write

Implements

ICommand

Inherited Members

Evaluation.expression , Evaluation.evaluatedExpression , Evaluation.varName , Evaluation.value , Evaluation.Expression , Evaluation.VarName , Evaluation.Value , Evaluation.Local , Evaluation.Compile() , Evaluation.ProcessExpression(string) , Command.IsDouble , Command.program , Command.parameterList , Command.parameters , Command.paramsint , Command.Program , Command.Name , Command.ParameterList , Command.Parameters , Command.Paramsint , Command.Set(StoredProgram, string) , Command.ProcessParameters(string) , Command.ToString() , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Methods

CheckParameters(string[])

Does nothing in this case.

```
public override void CheckParameters(string[] parameter)
```

Parameters

`parameter` string[]

Execute()

Write string to console. Also puts the evaluated string into this.evaluatedExpression so it can be used for outputting to other components, such as a text field.

```
public override void Execute()
```