Namespace BOOSE

Classes

[AboutBOOSE](#)

Basic Object Orientated Software Engineering.

This is a set of classes for implementing the BOOSE laguage.

To replace class functionality you can ether implement the relevant interface to completely replace all functionality, or extend the relevant class and override the interface methods, which will allow the calling of base methods in the original class.

Example of adding a command to BOOSE. Here I will call the new program AppBOOSE and append App to class names (but you can call them what you like).

step 1 extend ICanvas to have your new draw command that will be called when its command is executed, IAppCanvas.

step 2 extend this new interface with a class (AppCanvas) which creates a drawing object (bitmap?) does all the actual drawing.

step 3 extend appropriate command Interface to create a new command class. i.e. Rect(width, height) would extend ICommandTwoParameters.

step 4 extend the original factory if you want to use any of its existing commands (by calling base.MakeCommand() after looking for your new commands) or ICommandFactory if completely replacing it.

[BOOSEException](#)

Generic BOOSE Language exception Extends Exception

[Canvas](#)

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

[CanvasCommand](#)

Derived class to add a drawing surface that implements the ICanvas interface.

[CanvasException](#)

Exception generated by the Canvas class.

[Circle](#)

[Command](#)

Abstract class for commands with 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. CheckParameters

[CommandException](#)

    Exception genereted by the StoredProgram class.

[CommandFactory](#)

[CommandOneParameter](#)

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

[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 ELSE and ENDS

[ConditionalCommand](#)

    Extends Var because it has an expression, only this time it directly affects execution.

[DrawTo](#)

[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".

[FactoryException](#)

    exception genereted by the StoredProgram class

[For](#)

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

[If](#)

[MoveTo](#)

[Parser](#)

[ParserException](#)
    exception genereted by the StoredProgram class

[PenColour](#)

[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. Don't confuse with the class Program that a form creates with the main method in it

[StoredProgramException](#)
    exception genereted by the StoredProgram class

[Var](#)

[VarException](#)

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

Interfaces

[ICanvas](#)
    Use 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();

[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

Class AboutBOOSE

Namespace: [BOOSE](#)

Assembly: BOOSE.dll

Basic Object Orientated Software Engineering.
This is a set of classes for implementing the BOOSE language.
To replace class functionality you can ether implement the relevant interface to completely replace all functionality, or extend the relevant class and override the interface methods, which will allow the calling of base methods in the original class.

Example of adding a command to BOOSE. Here I will call the new program AppBOOSE and append App to class names (but you can call them what you like).
step 1 extend ICanvas to have your new draw command that will be called when its command is executed, IAppCanvas.
step 2 extend this new interface with a class (AppCanvas) which creates a drawing object (bitmap?) does all the actual drawing.
step 3 extend appropriate command Interface to create a new command class. i.e. Rect(width, height) would extend ICommandTwoParameters.
step 4 extend the original factory if you want to use any of its existing commands (by calling base.MakeCommand() after looking for your new commands) or ICommandFactory if completely replacing it.

```
public 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()](#)

Methods

about()

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

Returns

[string](#)⧉

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](#), [StoredProgramException](#), [VarException](#)

Inherited Members

[Exception.GetBaseException()](#) , [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 Canvas

Namespace:

Assembly: BOOSE.dll

Abstract class that implements CanvasInterface. See CanvasInterface documentation for what to implement. This class show handle all the drawing on your system according to above ocumentation. 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.

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↗

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

Derived class to add a drawing surface that implements the ICanvas interface.

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

Inheritance

[object](#) ← [Command](#) ← CanvasCommand

Implements

[ICommand](#)

Derived

[CommandOneParameter](#)

Inherited Members

[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 generated by the Canvas class.

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

Inheritance

[object](#) ← [Exception](#) ← [BOOSEException](#) ← CanvasException

Implements

[ISerializable](#)

Inherited Members

[Exception.GetBaseException()](#) , [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 Circle

Namespace: BOOSE

Assembly: BOOSE.dll

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

Inheritance

object ← Command ← CanvasCommand ← CommandOneParameter ← Circle

Implements

ICommand

Inherited Members

CommandOneParameter.param1 , CommandOneParameter.param1unprocessed ,
CanvasCommand.yPos , CanvasCommand.xPos , CanvasCommand.canvas , 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.MemberwiseClone() , 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 commands with 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. CheckParameters

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

Inheritance

[object](#) ← Command

Implements

[ICommand](#)

Derived

[CanvasCommand](#), [Var](#)

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()
```

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 orf 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:

Assembly: BOOSE.dll

Exception genereted by the StoredProgram class.

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

Inheritance

object ← Exception ← BOOSEException ← CommandException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , 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. Currentlt creates moveto, drawto, circle, pen and var". To create additional commands create a Factory that iextends 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 two parameters. Each command has an Xpos and Ypos that operates from the current cursor position.

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

Inheritance

[object](#) ← [Command](#) ← [CanvasCommand](#) ← CommandOneParameter

Implements

[ICommand](#)

Derived

[Circle](#), [CommandTwoParameters](#)

Inherited Members

[CanvasCommand.yPos](#) , [CanvasCommand.xPos](#) , [CanvasCommand.canvas](#) , [CanvasCommand.Canvas](#) , [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.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.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 ELSE and ENDS

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

Inheritance

[object](#) ← [Command](#) ← [Var](#) ← [ConditionalCommand](#) ← CompoundCommand

Implements

[ICommand](#)

Derived

[Else](#), [End](#)

Inherited Members

[ConditionalCommand.EndLineNumber](#) , [ConditionalCommand.Condition](#) ,
[ConditionalCommand.LineNumber](#) , [ConditionalCommand.CondType](#) , [ConditionalCommand.Execute()](#) ,
[Var.Expression](#) , [Var.VarName](#) , [Var.Value](#) , [Var.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.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

Properties

CorrespondingCommand

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

Property Value

[ConditionalCommand](#)

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()

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

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

Exceptions

VarException

Class ConditionalCommand

Namespace: [BOOSE](#)

Assembly: BOOSE.dll

Extends Var because it has an expression, only this time it directly affects execution.

```
public class ConditionalCommand : Var, ICommand
```

Inheritance

[object](#) ← [Command](#) ← [Var](#) ← ConditionalCommand

Implements

[ICommand](#)

Derived

[CompoundCommand](#), [For](#), [If](#), [While](#)

Inherited Members

[Var.Expression](#) , [Var.VarName](#) , [Var.Value](#) , [Var.CheckParameters(string[])](#) , [Var.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.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

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](⧉)

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 corrensponding "end" command.

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

Exceptions

CommandException

Enum ConditionalCommand.conditionalTypes

Namespace: BOOSE

Assembly: BOOSE.dll

```
public enum ConditionalCommand.conditionalTypes
```

Fields

commFor = 2

commIF = 0

commWhile = 1

Class DrawTo

Namespace: BOOSE

Assembly: BOOSE.dll

```
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.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](#) ← [Var](#) ← [ConditionalCommand](#) ← [CompoundCommand](#) ← Else

Implements

[ICommand](#)

Inherited Members

[CompoundCommand.CorrespondingCommand](#) , [ConditionalCommand.EndLineNumber](#) , [ConditionalCommand.Condition](#) , [ConditionalCommand.LineNumber](#) , [ConditionalCommand.CondType](#) , [Var.Expression](#) , [Var.VarName](#) , [Var.Value](#) , [Var.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.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

Properties

CorrespondingEnd

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

Property Value

[End](#)

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()

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 corrensponding "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".

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

Inheritance

[object](#) ← [Command](#) ← [Var](#) ← [ConditionalCommand](#) ← [CompoundCommand](#) ← End

Implements

[ICommand](#)

Inherited Members

[CompoundCommand.CorrespondingCommand](#) , [CompoundCommand.CheckParameters(string[])](#) , [ConditionalCommand.EndLineNumber](#) , [ConditionalCommand.Condition](#) , [ConditionalCommand.LineNumber](#) , [ConditionalCommand.CondType](#) , [Var.Expression](#) , [Var.VarName](#) , [Var.Value](#) , [Var.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.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

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

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

Inheritance

[object](#) ← [Command](#) ← [Var](#) ← [ConditionalCommand](#) ← For

Implements

[ICommand](#)

Inherited Members

[ConditionalCommand.EndLineNumber](#) , [ConditionalCommand.Condition](#) ,
[ConditionalCommand.LineNumber](#) , [ConditionalCommand.CondType](#) , [Var.Expression](#) , [Var.VarName](#) ,
[Var.Value](#) , [Var.CheckParameters(string[])](#) , [Var.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.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

Properties

From

WHere counting starts.

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

Property Value

[int](#)

LoopControlV

For loop uses a variable to do its counting.

```
public Var LoopControlV { get; }
```

Property Value

[Var](#)

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 teh 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

Use 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.

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⧉

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[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.

```csharp
public interface ICommand
```

Methods

CheckParameters(string[])

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

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

Parameters

Parameters [string](#) []

Compile()

Called before the command is run.

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

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

Parameters

program string⬀

   big string of a complete program seperated by newlines

Returns

string⬀

   Error list as one long string

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(Var)

Add a Var object to the StoredProgram.

```
void AddVariable(Var Variable)
```

Parameters

Variable [Var](Var)

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 interger 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.

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

Parameters

varName [string ↗](#)

Returns

[int ↗](#)

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:

Assembly: BOOSE.dll

```
public class If : ConditionalCommand, ICommand
```

Inheritance

object ⇄ ← Command ← Var ← ConditionalCommand ← If

Implements

ICommand

Inherited Members

ConditionalCommand.EndLineNumber , ConditionalCommand.Condition ,
ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.Compile() ,
ConditionalCommand.Execute() , Var.Expression , Var.VarName , Var.Value ,
Var.CheckParameters(string[]) , Var.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.MemberwiseClone() ⇄ , object.ReferenceEquals(object, object) ⇄

Class MoveTo

Namespace: [BOOSE](#)

Assembly: BOOSE.dll

```
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.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

```
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.

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.

```
public virtual 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 entire program, adding valid command objects to the StoredProgram. An errorlist string is generated containing any systax error messages.

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

Parameters

`program` string⧉

   big string of a complete program seperated by newlines

Returns

string⧉

   Error list as one long string

Class ParserException

Namespace:

Assembly: BOOSE.dll

exception genereted by the StoredProgram class

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

Inheritance

object ← Exception ← BOOSEException ← ParserException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , 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 PenColour

Namespace: [BOOSE](#)

Assembly: BOOSE.dll

```
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.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 orf parameters and throw a CommandException if not.

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

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. Don't confuse with the class Program that a form creates with the main method in it

```
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)

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

Parameters

canvas ICanvas

Fields

INFINITE_LOOP_THRESHOLD

```
public const int INFINITE_LOOP_THRESHOLD = 500
```

Field Value

[int](int)↗

SyntaxOk

```
public bool SyntaxOk
```

Field Value

bool↗

TYPICAL_LOOP_SIZE

```
public const int TYPICAL_LOOP_SIZE = 20
```

Field Value

[int ↗](#)

Properties

PC

property for int pc the program counter

```
public 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.

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

Parameters

C [Command](#)

Returns

[int ↗](#)

    index at which member was added

AddVariable(Var)

Add a Var object to the StoredProgram.

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

Parameters

`Variable` [Var](#)

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 result as a String

```
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.

FindVariable(Var)

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

```
public int FindVariable(Var Variable)
```

Parameters

Variable [Var](#)

Returns

[int](#)⟐

    Position or -1 if not found

FindVariable(string)

Get a variables position in the Variables table arraylist

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

Parameters

varName string↗

Returns

int↗

-1 if not found, otherwise the position.

GetVarValue(string)

Returns a variables value from it string name.

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

Parameters

varName string↗

Variable name.

Returns

int↗

Integer value of variable.

Exceptions

[StoredProgramException](#)

If variable is not found.

GetVariable(int)

Return the Variable object.

```
public Var GetVariable(int index)
```

Parameters

index int☐

    Position in Variables table to extract.

Returns

[Var](#)

    Position of object in Variables Table.

Exceptions

[StoredProgramException](#)

    Thrown if invalid index is passed.

IsExpression(string)

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

Parameters

expression string☐

Returns

bool☐

Pop()

```
public ConditionalCommand Pop()
```

Returns

[ConditionalCommand](#)

Push(ConditionalCommand)

Push a compund command onto the stack

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

Parameters

Com ConditionalCommand

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.

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

Exceptions

StoredProgramException

UpdateVariable(string, int)

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

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

Parameters

varName string

String nme 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.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 Var

Namespace: [BOOSE](#)

Assembly: BOOSE.dll

```
public class Var : Command, ICommand
```

Inheritance

[object](#) ← [Command](#) ← Var

Implements

[ICommand](#)

Derived

[ConditionalCommand](#)

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.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

Constructors

Var()

Value of variable once the expression has been calculated.

```
public Var()
```

Properties

Expression

The right hand side of an expression.

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

Property Value

[string ⧉](#)

Value

```
public 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, in this case Var.

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

Property Value

[string ⧉](#)

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[] parameterList)
```

Parameters

parameterList 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 orf parameters and throw a CommandException if not.

```
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

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

Parameters

Expression string⧉

Returns

string⧉

S

Exceptions

VarException

Class VarException

Namespace:

Assembly: BOOSE.dll

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

Inheritance

object ← Exception ← BOOSEException ← VarException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , 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 class While : ConditionalCommand, ICommand
```

Inheritance

object ← Command ← Var ← ConditionalCommand ← While

Implements

ICommand

Inherited Members

ConditionalCommand.EndLineNumber , ConditionalCommand.Condition , ConditionalCommand.LineNumber , ConditionalCommand.CondType , ConditionalCommand.Compile() , ConditionalCommand.Execute() , Var.Expression , Var.VarName , Var.Value , Var.CheckParameters(string[]) , Var.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.MemberwiseClone() , object.ReferenceEquals(object, object)