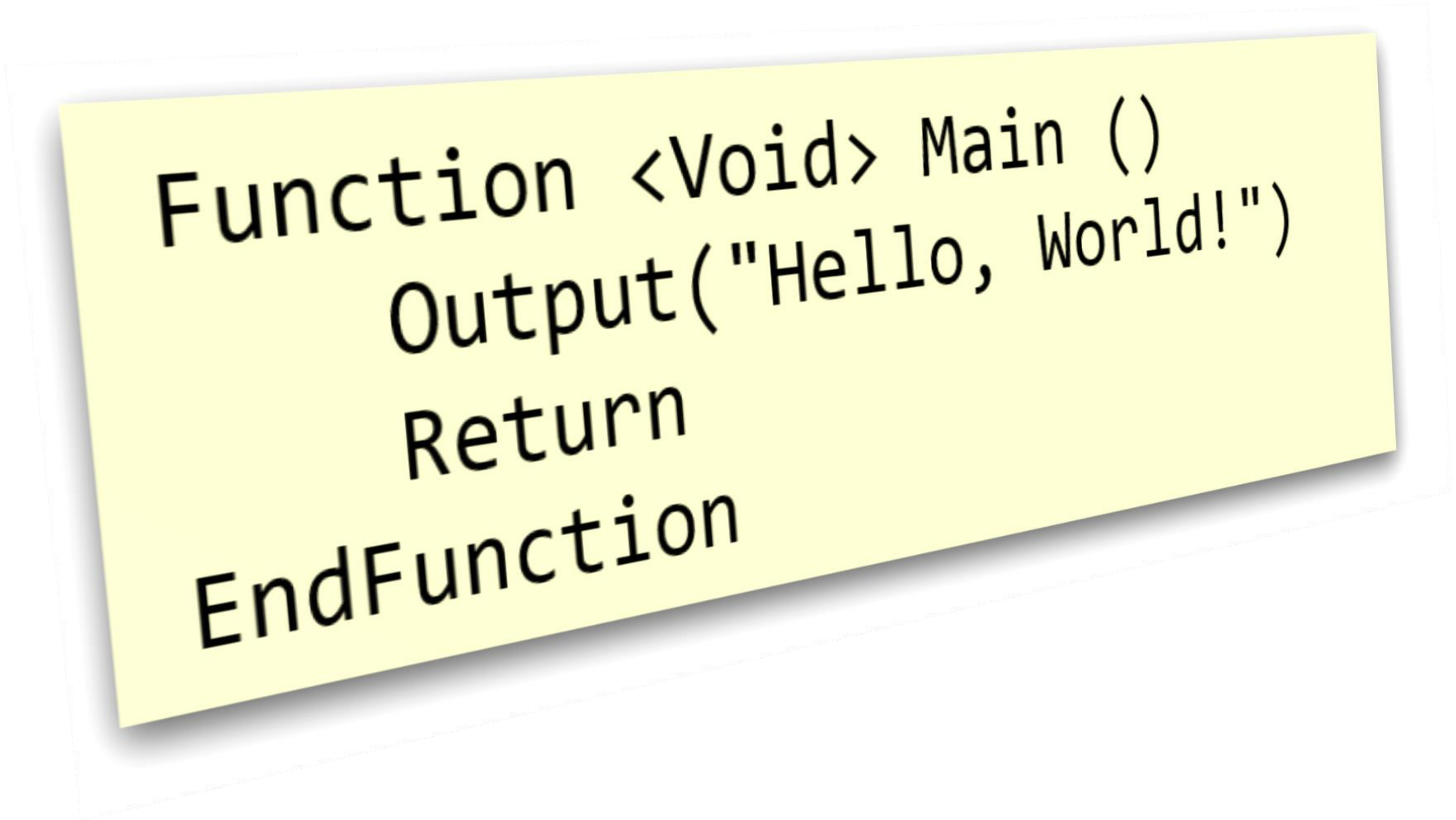
DocScript



```
Function <Void> Main ()
    Output("Hello, World!")
    Return
EndFunction
```

6 Keywords

3 DataTypes

15 Operators

3 Implementations

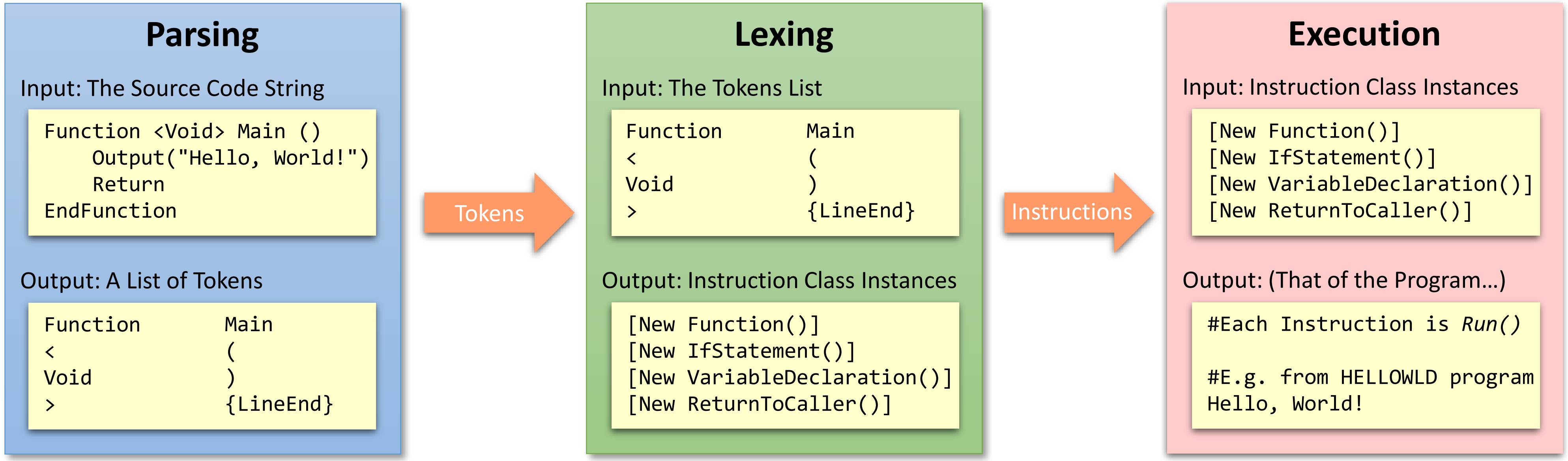
Example

Cutting to the chase: a sample of DocScript Source

```
Function <Number> Main (<String@> _CLAs)

    #Input looks like {"-Name", "Ben", "-Age", "13"}
    #Get the Value for an Input()'ed Key

    <String> _Key : Input("Argument Key:")
    <String> _Value : GetCLAValueFromKey(_Key)

    Output("Value: " & _Value)
    Return 0

EndFunction

Function <String> GetCLAValueFromKey(<String@> _CLAs, <String> _Key)

    <Number> _CurrentCLAIndex : 0

    While (LessThan(_CurrentCLAIndex, [Array_MaxIndex(_CLAs) + 1]))
        If (Array_At(_CLAs, _CurrentCLAIndex) = ["-" & _Key])
            Return StringArray_At(_CLAs, _CurrentCLAIndex + 1)
        EndIf
        CurrentCLAIndex : [CurrentCLAIndex + 1]
    EndWhile

    Return "No Value found for Key [" & _Key & "]"

EndFunction
```

Interpretation Proceß

Three stages in understanding and executing a DocScript Program



Parsing

Input: The Source Code String

```
Function <Void> Main ()
    Output("Hello, World!")
    Return
EndFunction
```

Output: A List of Tokens

```
Function        Main
<               (
Void            )
>               {LineEnd}
```

Lexing

Input: The Tokens List

```
Function        Main
<               (
Void            )
>               {LineEnd}
```

Output: Instruction Class Instances

```
[New Function()]
[New IfStatement()]
[New VariableDeclaration()]
[New ReturnToCaller()]
```

Execution

Input: Instruction Class Instances

```
[New Function()]
[New IfStatement()]
[New VariableDeclaration()]
[New ReturnToCaller()]
```

Output: (That of the Program…)

```
#Each Instruction is Run()

#E.g. from HELLOWLD program
Hello, World!
```

Any comments which appear in the source (e.g.
`#This is a Comment`
) do not become tokens. They are discounted at this first stage.

The Lexing occurs in the constructors for the various [IInstruction]s. When invoked, these constructors are passed the part of the TokenList required to create the [IInstruction].

The [Program] object may contain several [GlobalVariableDeclaration]s, which are executed first. Next, the Main() [Function] is executed, after which the program has finished.

```
Steps:                DocScript.Runtime.Parser.GetTokensFromSource()
REM ╔══════════════════════════════════════════╗
REM ║         DocScript Parsing Process        ║
REM ╚══════════════════════════════════════════╝

REM 1) Initialisation
'- Ensure all LineBreaks are valid (CrLf)
'- Load in Lines of Source

REM 2) Segmentation
'- Blank out any #Comments or Whitespace Lines
'- Ensure nothing already exists in the Source which matches the SLIT RegExp
'- Replace StringLiterals with SLITs e.g. $SLIT_0$
'- Generate Segmented Tokens:
'- (For Each Line, and For Each Character thereof, evaluate if it's a WordChar or SplitAtChar...)
'- Remove any Null Tokens (Whitespace, etc...)
'- Ensure all remaining characters are valid (E.g. No SpeechMarks) (Best to do this now as we have the [TokenLocation]s)
'- Replace any SLITs with their original StringLiterals

REM 3) Classification
'- For Each Token, attempt to match it to a RegExp for its TokenType
'- Ensure all Bracket usage is balenced (Best to do this now as we have the TokenTypes for easy GrammarChar filtering)

REM [_RawSourceLines] → [_CleanSourceLines] → [_SegmentedTokens] → [_NonNullTokens] → [_TokensWithStringLiterals] →
[_ClassifiedTokens]
```

```
REM ╔════════════════════════════════════════╗
REM ║  DocScript Expr. Tree Construction Process  ║
REM ╚════════════════════════════════════════╝

REM 1) Initial Validation
'                              - Ensure _RawTokens is not Empty
'                              - Reassign _RawTokens to not end in a [LineEnd] Token if it currently does
'                              - Ensure _RawTokens all have a permitted TokenType
'                              - Ensure each opening bracket "(" or "[" has a corrosponding closing bracket...
'                              ... (even though we know the brackets for the source as a whole are balenced)
REM 2) LBL Production
'                              - Produce the Top-Level LBL (Linear Bracketed Level)
'                              - Simplify this LBL into an unambigous form
'                              - Validate this simplified LBL to ensure that the expression is well-formed
REM 3) IOT Collapsing
'                              - Identify the Indexes and Prescedances, of Operators in LBL
'                              - Order this OperatorsList by the Presecedance and Associativity of the operators
'                              - Starting with the highest-prescedance Operator, collapse the LBL into IOTs (Intermediate Operator Trees)
'                              - Assemble these IOTs into the RootTreeNode, via the SCIs (Scanned Component Indicators) of each LooseOperatorExpr

REM [_RawTokens] → [_TopLevelLBL] → [_SimplifiedLBL] → [_ExprTreeRoot]
```

```
REM ╔══════════════════════════════════════════╗
REM ║        DocScript Execution Process       ║
REM ╚══════════════════════════════════════════╝

REM 1) Initialisation
'- If the ExeCxt is Nothing, then this Program instance shouldn't Run()
'- Generate the GlobalSymbolTable
'- Add Program Functions to the GlobalSymbolTable

REM 2) Invocation
'- Execute() each GlobalVarDec
'- Run() DSFunction Main (located in the GlobalSymbolTable)
'                              - If it is the Signature with the CLAs, then:
'                                                          - Pass in _CommandLineArguments$()
'                                                          - Ensure that a ReturnValue is produced
'- Update the GlobalSymbolTable, to include any Modifications from during execution
'- Return this Program's ExitCode, wrapped in an ExeRes
```

Syntax

Physical form and grammar used in DocScript Source

Assignment Syntax:
<DataType> Identifier : Expression
Or
Identifier : Expression

Keywords:

If	EndIf	Else
While	EndWhile	
Loop	EndLoop	
Return		
Function	EndFunction	

()	Function Declarations and Calls	Main()
<>	DataType Specification	<String> Name
[]	Expression Subdivision	[5 + 3] / 2
{}	(Reserved)	* (Possibly ArrayLiterals)

The ambiguity is reduced by using different characters for the Function Calls and Expression Subdivision.

Notable Reserved Chars:

.	Numeric Literals
_	Numeric Literals
:	Assignment
#	Comments
$	Parser Internal use

General Rules:

- The language is NOT case-sensitive (apart from inside StringLiterals)
- Tabs have no semantic significance
- Comments can only befall at the start of a line, not mid-way through one
- The Character $ is reserved for internal use by the Parser and is not valid as an Identifier, Operator, or Grammar Char.
- Statement Ends now look like “EndFunction” not “End-Function”
- Identifiers may ONLY contain [A-Za-z_]

Token Types:

1. Unresolved
2. StringLiteral
3. NumericLiteral
4. BooleanLiteral
5. Keyword
6. DataType
7. Identifier
8. DSOperator
9. GrammarChar
10. LineEnd
11. StatementEnd

Instruction Types:

VariableDeclaration	<String> Name	<String> Name = "Andrew"
VariableAssignment	Name = "Ryan"	
ReturnToCaller	Return	Return "Value"
FunctionCall	SayHello()	
FunctionDeclaration*	Function <Void> SayHello ()	
IfStatment	If (True)	
WhileStatment	While (True)	
LoopStatment	Loop (10)	

* = Cannot appear within a Function Body

Whereas an [Instruction] ends at the next {LineEnd}, a [Statement] ends at its StatementEnd line.

Possible LineTypes:

```
#Comment
<String> Name
<String> Name : "Ben"
Name : "Ben"
SayHello()
SayHello(Name)
Return
Return 0
Function <Void> SayHello ()*
Function <Void> SayHello (<String> Name)*
EndFunction*
If (True)
Else
EndIf
While (True)
EndWhile
Loop (10)
EndLoop
```

* = Cannot appear within a Function Body

Operators

Built-in useful logic used in expressions along with their Operands

Overloading

= when the same operator notation has multiple has multiple logical definitions and can therefore do different things, depending on how it appears syntactically.

Example

```
# Subtraction:
5 – 1
# Polarity Inversion:
-9
```

Associativity

= when operators of the same Precedence appear in the same bracketed level, the Associativity governs whether to evaluate from left to right or vice versa. Left-associative means from left-to-right.

Example

```
# Left-Associative
<Number> A = 96 / 8 / 4
# Same as:
<Number> B = (96 / 8) / 4
```

Precedence

= when one operator is executed before another different operator, despite the two being on the same bracketed level. Each operator has it’s own precedence. Operators with a *higher* precedence are executed first.

Example

```
5 + 3 * 2
#Answer: 11
( 5 + 3 ) * 2
#Answer: 16
```

Commutativity

= when the operands of an operator can be swapped, without changing the result.

Example

```
8 * 4
#Answer: 32
4 * 8
#Answer: Still 32
```

Monadic, Dyadic, Triadic:	*The fact that there are {1, 2, or 3} states available*
Unary, Binary, Ternary:	*The fact that something is in one of {1, 2, or 3} separate states*

The Logical Operators are Short-circuiting
All DocScript Operators are left-associative.
All Unary Operators have their Operand to their Right
All Unary Operators must have the highest precedence
< and > are NOT used as Operators, as they are the DataTypeBrackets

The Operators are Assign (:) and the Expr. Operators...

[] Expression Operators

Higher=First

Operator	Description	Operands Type	Return Type	Precedence
=	Equality Comparison	2 Anything	Boolean	1
&	Concatenation	2 String	String	2
¬	Logical Not	1 Boolean	Boolean	8
'	Logical And	2 Boolean	Boolean	7
|	Logical Or	2 Boolean	Boolean	5
¦	Logical Xor	2 Boolean	Boolean	6
+	Addition	2 Number	Number	4
-	Subtraction	2 Number	Number	4
*	Multiplication	2 Number	Number	3
/	Division	2 Number	Number	3
^	Exponentiation	2 Number	Number	3
%	Modulo	2 Number	Number	3
~	Invert Polarity	1 Number	Number	8
	Spare Chars: \`!£?;			

Functions

Executable blocks of script which can Return a value

There are *two* valid EntryPoint Declarations

```
Function <Number> Main (<String@> _CLAs)
```

```
Function <Void> Main ()
```

The simplest-possible DocScript Program

```
Function <Void> Main ()
        Return
EndFunction
```

Instruction → Execute()
Program → Run()

Possible LineTypes for within a Function:

```
1.  #Comment
2.  <Number> Age
3.  <Number> Age = 17            E
4.  Age = 17 + 1                 E
5.  Return
6.  Return "Value"               E
7.  SayHello()
8.  SayHello("Ben")              E
9.  If (True)                    E
10. Else
11. End-If
12. While (True)                 E
13. End-While
14. Loop (10)                    E
15. End-Loop

E = Includes an Expression (Expr.)
```

General Rules:

- All Arguments are passed by Value, not Referance
- It is the responsibility of a DSFunction to append the LocalSymbolTable with the Arguments when Execute() is called
- All Arguments (IExpression) are fully resolved to (IDataValue)s before a function is executed

FunctionCall	*The IInstruciton representing a call to a DSFunction*
FunctionCallExpr	*The IExpression representing a call to a DSFunctino and the Return Value thereof*
DSFunction	*The Object representing a Function as declared in source*

Delegate Function BuiltInFunction(ByRef _ExeCxt, ByRef _SymTbls, ByVal _Arguments) As IExpression

*[*] means * occours indirectly; downstream of the current Iinstruction (E.g. Inside an Expression or the Contents of an IStatement)*

IInstruction Implementation	Example	Required SymbolTableStack Access
FunctionCall	Output("Hello, World!")	Global: Read; [Global: Read/Write]; [Locals: Read]
ReturnToCaller	Return "Hello"	[Global: Read/Write]; [Locals: Read]
VariableAssignment	Name = "Ben"	Global: Read/Write; Locals: Read/Write; [Global: Read/Write]; [Locals: Read]
VariableDeclaration	<String> Name = "Ben"	*
DSFunction	Function <Void> Main () …	*
IfStatement	If (A | ¬B) …	*
LoopStatement	Loop (10) …	*
WhileStatement	While (¬False) …	*

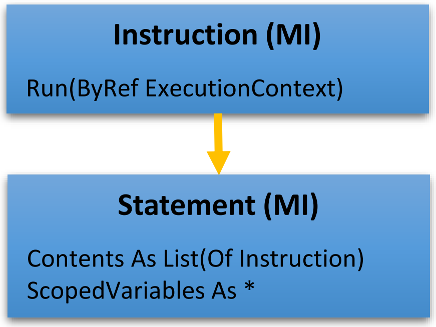
Instructions

Individual executable components

Possible LineTypes for within a Function:

```
1.   #Comment
2.   <Number> Age
3.   <Number> Age : 17     E
4.   Age : 17 + 1          E
5.   Return
6.   Return "Value"        E
7.   SayHello()
8.   SayHello("Ben")       E
9.   If (True)             E
10.  Else
11.  EndIf
12.  While (True)          E
13.  EndWhile
14.  Loop (10)             E
15.  EndLoop

E = Includes an Expression (Expr.)
```

```
- Instruction : Run()
          VariableDeclaration
          VariableAssignment
          ReturnToCaller
          FunctionCall
          - Statement : Contents
                DSFunction
                IfStatement
                WhileStatement
                LoopStatement
```



Instruction (MI)

Run(ByRef ExecutionContext)

Statement (MI)

Contents As List(Of Instruction)
ScopedVariables As *

Parameter

```
<String@> _CLAs

Identifier As String
DataType As Type
```

VariableDeclaration

```
<String> Name
<String> Name = "Ben" & ToString(7)

Identifier As String
DataType As Type
DeclarationType As VarDecType {DeclareOnly, DeclareAndAssign}
InitialiserExpression As Expression
```

VariableAssignment

```
Name = "Ben" & ToString(7)

Identifier As String
InitialiserExpression As Expression
```

ReturnToCaller

```
Return
Return "Value"

ReturnType As ReturnInstructionType {ReturnOnly, ReturnWithValue}
ReturnValue As Expression
```

FunctionCall

```
Output()
Output("Value")

Identifier As String
Arguments As List(Of Expression)
```

DSFunction

```
Function <Void> SayHello ()
Function <Number> Main (<String@> _CLAs)

Identifier As String
ReturnType As Type
Arguments As List(Of Parameter)
```

Contents As List(Of Instruction)
ScopedVariables As *

IfStatement

```
If (True)

Condition As Expression
```

WhileStatement

```
While (True)

Condition As Expression
```

LoopStatement

```
Loop (10)

LoopCount As Expression
```

```
'Whereas an IExpression represents an UNRESOLVED Tree which *can* produce a value...
'...an IDataValue represents a RESOLVED Datum which is itself a value.
'IExpressions DON'T have a DocScript DataType, whereas IDataValues DO.
```

ReturnToCaller needs a way of communicating back up to the parent Function – Return an ExecutionResult Object?

Parameter = Placeholder
Argument = Expression

Data Types [0]

Tags for data to indicate their value type

Valid Literals:

Number	10	10.0	10_10
String	"String"		
Boolean	True	False	

- All Variables have a default value:

• <Number>	0
• <String>	""
• <Boolean>	False
• <*@>	(Empty Array)

Number	/@
String	/@
Boolean	/@
Void	

Local SymbolTable

Identifier	EntryType	DataType	Value
_Names	Variable	String@	{"One", "Two"}
_Age	Variable	Number	101_2
_IsGood	Constant*	Boolean	True
_CLAs	Argument	String@	{}

- Variables and Arguments only
- One Local SymTbl per Istatement (DSFunction, IfStatement, …)

Global SymbolTable

Identifier	EntryType	DataType	Value
Main	DSFunction	Number	{DSFunction}
Output	BuiltInFunction	Void	{BuiltInFunction}
Names	Variable	String@	{"One", "Two"}
Age	Variable	Number	101_2
IsGood	Constant*	Boolean	True

- DSFunctions, BuiltInFunctions and Variables Only
- One Global SymTbl per Program

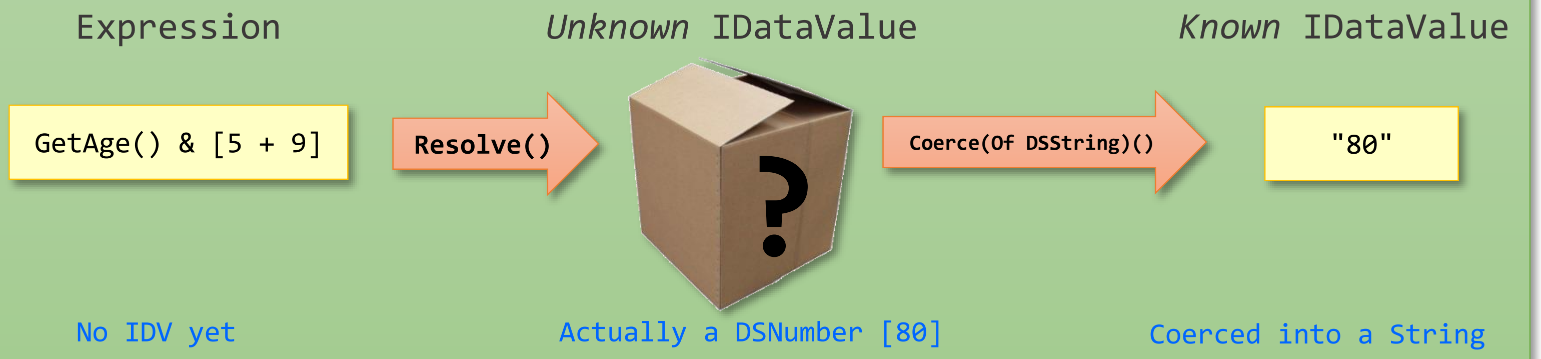
**** = Might be added in future***

During Expression Construction, the ReturnTypes and Argument Types of Functions are not known (well – they're not looked up anyway). Only the Literals can have their values resolved.

Variable Lookup Order: Localmost → FunctionLocal → Global
Function Lookup Order: Global (Program → BuiltIn's)

Numeric Literal can have up to 4 decimal places

DocScript Type Coercion (DS-TC)



Reminder: The IDataValue Interface is implemented by DSString, DSNumber, DSBoolean, and DSArray(Of * As {IDataValue}). Each of these types is a simple wrapper for a tangible .NET type: (respectively) String, Number, Boolean, and List(Of *).

Data Types [1]

Tags for data to indicate their value type

DocScript Type Coercion (DS-TC)

Target ↓

Input →

	String	Number	Boolean	String@	Number@	Boolean@
String	No Change	Try/Parse()	Refuse	Refuse (?)	Refuse (?)	Refuse (?)
Number	ToString()	No Change	Refuse	Refuse (?)	Refuse (?)	Refuse (?)
Boolean	ToString()	Refuse	No Change	Refuse (?)	Refuse (?)	Refuse (?)
String@	ToString()	Refuse	Refuse	No Change	Refuse (?)	Refuse (?)
Number@	ToString()	Refuse	Refuse	Refuse (?)	No Change	Refuse (?)
Boolean@	ToString()	Refuse	Refuse	Refuse (?)	Refuse (?)	No Change

Key:

- Blue → Input is already in Target type
- Green → Can almost certainly derive Target from Input
- Orange → Might be able to derive Target from Input
- Red → DS-TC will refuse to produce an output of the Target type, from the Input

Expressions

Resolvable collections of Operators, Literals, Variables, and FunctionCalls, which produce a value

Realisations:

- Resolve()ing an expression requires all the same resources as Execute()ing an Iinstruction; Symbol Tables are needed for Variable and Function Lookups
- We don’t actually need to know what the value of any of the expression components are, for the purposes of constructing the Expression Tree. All we care about at this stage is which token is an Operator and which a Literal or a Variable etc…
- The operators with the lowest precedence will be the highest-up in the Tree

ExpressionTree Construction Process

```
Raw:     5           + 9        - [GetAge() ^ 1101_2]        * Take(Age)
LBL:     Lit         + Lit      - [BracketedExpr (3)]        * [FunctionCallExpr (1)]
IOT0:    Lit         + Lit      - [OperatorExpr]
IOT1:    [OperatorExpr]         - [OperatorExpr]
IOT2:    [OperatorExpr]
```

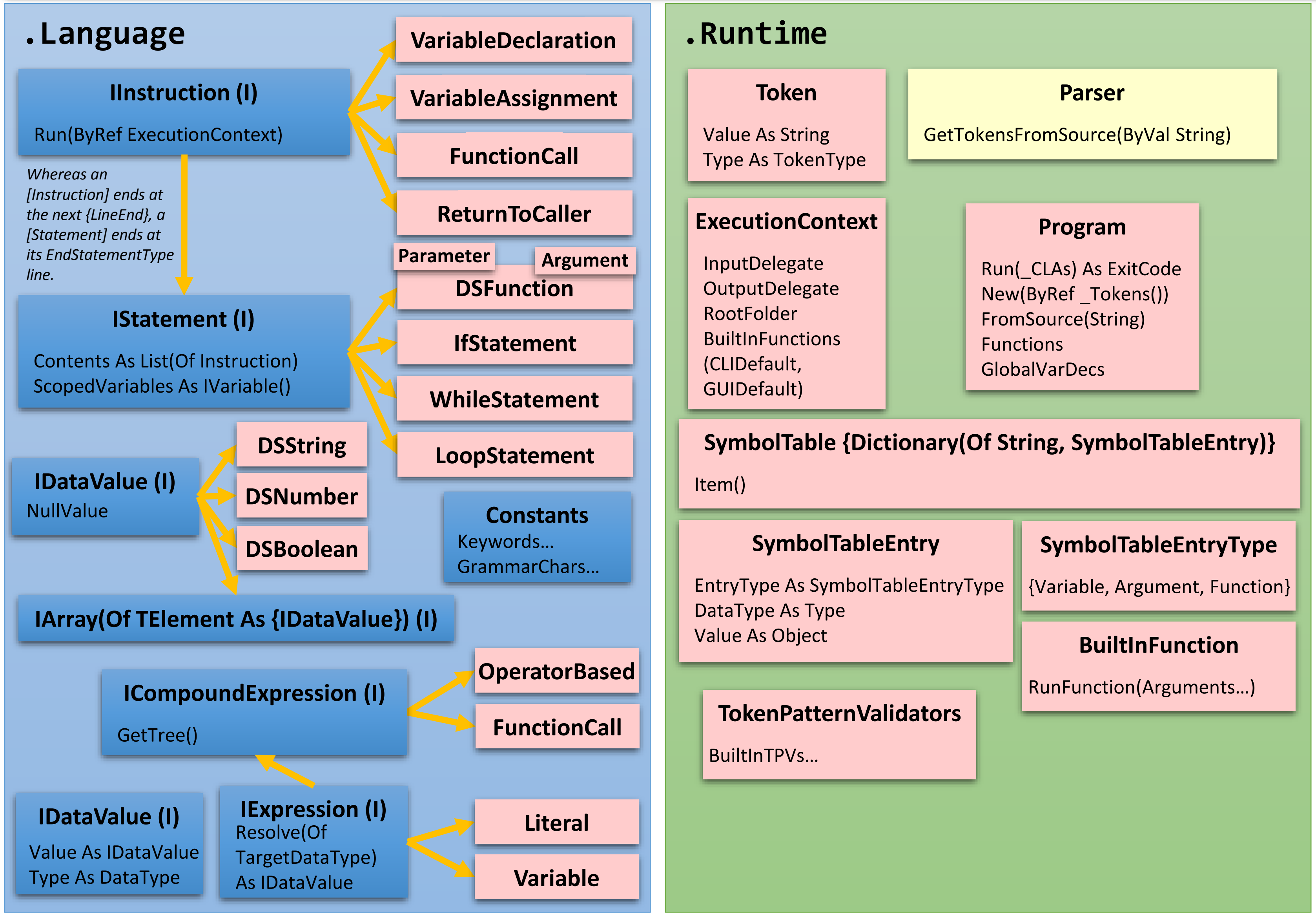
Explainations

With a ***Linear Bracketed Level (LBL)*** constructed, we only need to worry about the operators and their prescedance, because we *know*, that the contents of the BracketedExprs and FunctionCall Arguments must be resolved first. Collapsing to ***Intermediate Operator Trees (IOTs)*** is the subsequent stage whereby the operators with the highest prescedance are the first to be collapsed into OperatorExprs. This eventually forms a Complete Tree, free from any of the LBL Placeholders (for the Operators, BracketedExprs, and FunctionCalls).

Namespaces

Logical segmentation in the DocScript Interpretation Engine (DSIE/DLL)

(MI) = MustInherit   (I) = Interface   Blue = BaseOnly   Pink = Class   Yellow = Module   Orange = Enum

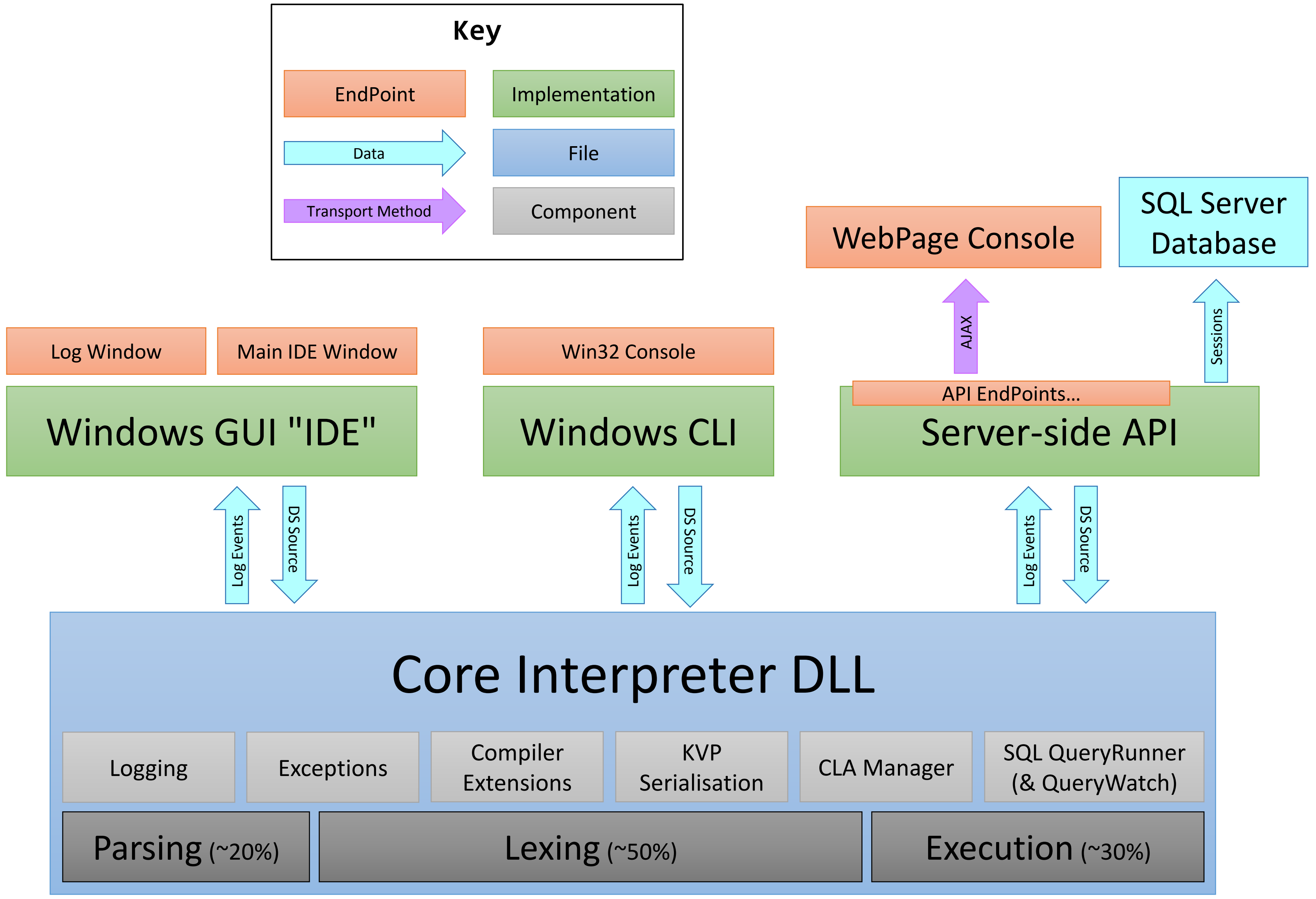


?????
DSFunction and BuiltInFunction do not use the Execute() Method to be executed. They use RunFunction(Arguments())

```
DocScript
        Language
                [Instruction] ... Run()
                [Statement ← Instruction]
        Runtime
                [ExecutionContext]
                [Program]
                [LogMessage]
                [Token]
                [SymbolTable]
```

Architecture

Interaction of the DocScript Interpretation and Implementation Components

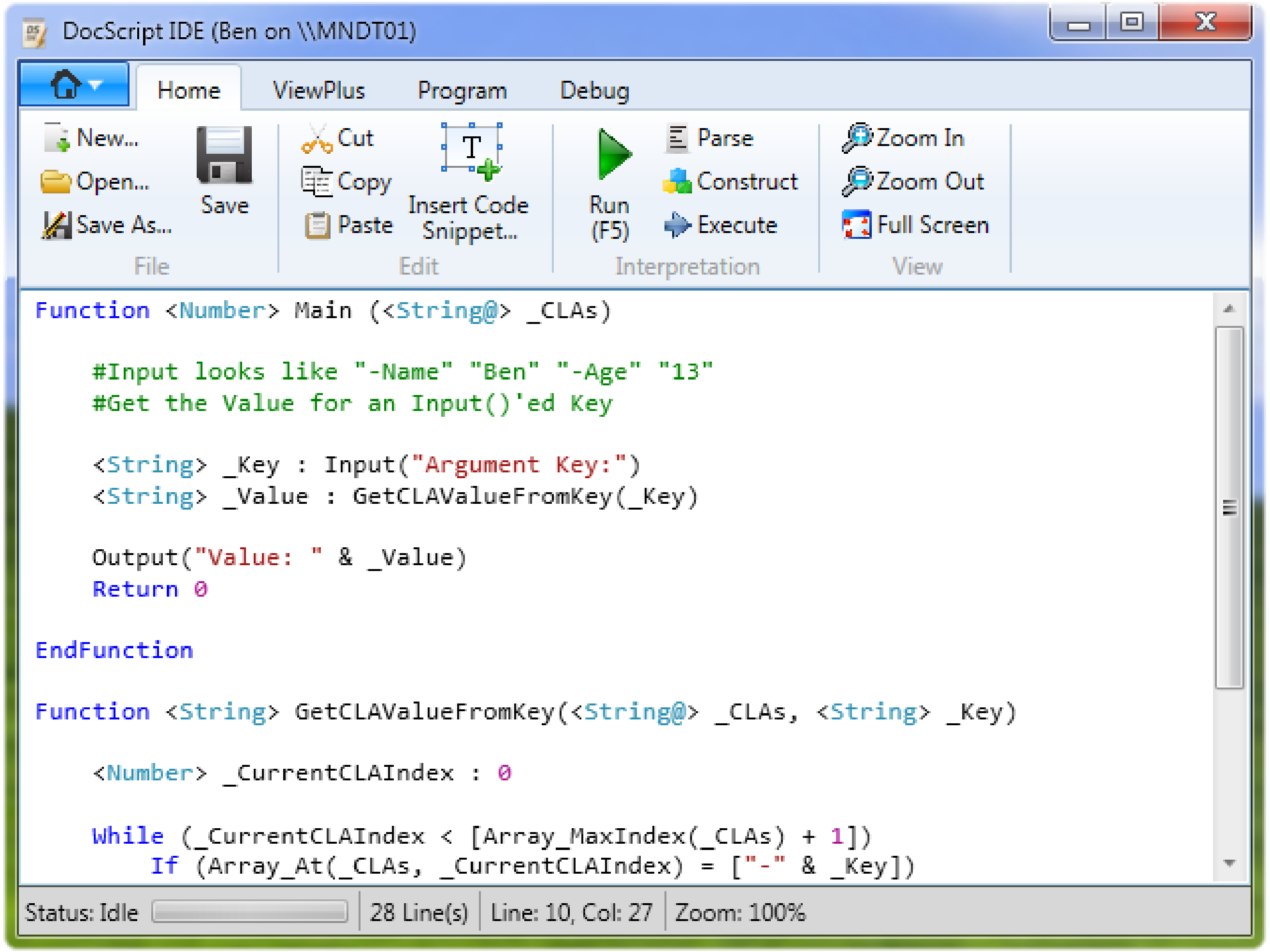


DocScript Windows IDE

Implementation of DSIE/DLL into a Graphical Windows script composition and debugging environment

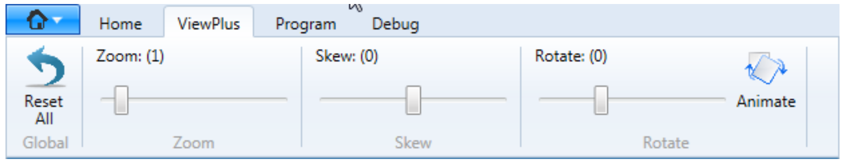
Possible Features

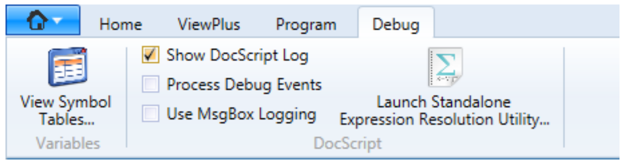
- Insert… Snippet,
- Expression Tree Plotter



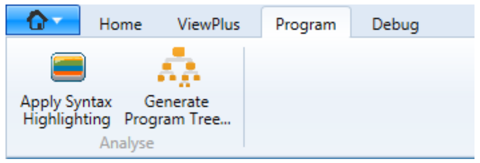
```
Function <Number> Main (<String@> _CLAs)

    #Input looks like "-Name" "Ben" "-Age" "13"
    #Get the Value for an Input()'ed Key

    <String> _Key : Input("Argument Key:")
    <String> _Value : GetCLAValueFromKey(_Key)

    Output("Value: " & _Value)
    Return 0

EndFunction

Function <String> GetCLAValueFromKey(<String@> _CLAs, <String> _Key)

    <Number> _CurrentCLAIndex : 0

    While (_CurrentCLAIndex < [Array_MaxIndex(_CLAs) + 1])
        If (Array_At(_CLAs, _CurrentCLAIndex) = ["-" & _Key])
```

Status: Idle | 28 Line(s) | Line: 10, Col: 27 | Zoom: 100%





✓ Show DocScript Log
☐ Process Debug Events
☐ Use MsgBox Logging



OM: Add some Orange
17122022: (Done)

Special Colour for BIFs?

Token Colours ↓

TokenType	Colour
Comment	#Comment
Unresolved	*
StringLiteral	"Value"
NumericLiteral	0110_2
BooleanLiteral	True
Keyword	Function
DataType	Number
Identifier	Main
DSOperator	&
GrammarChar	(
LineEnd	*
StatementEnd	EndFunction

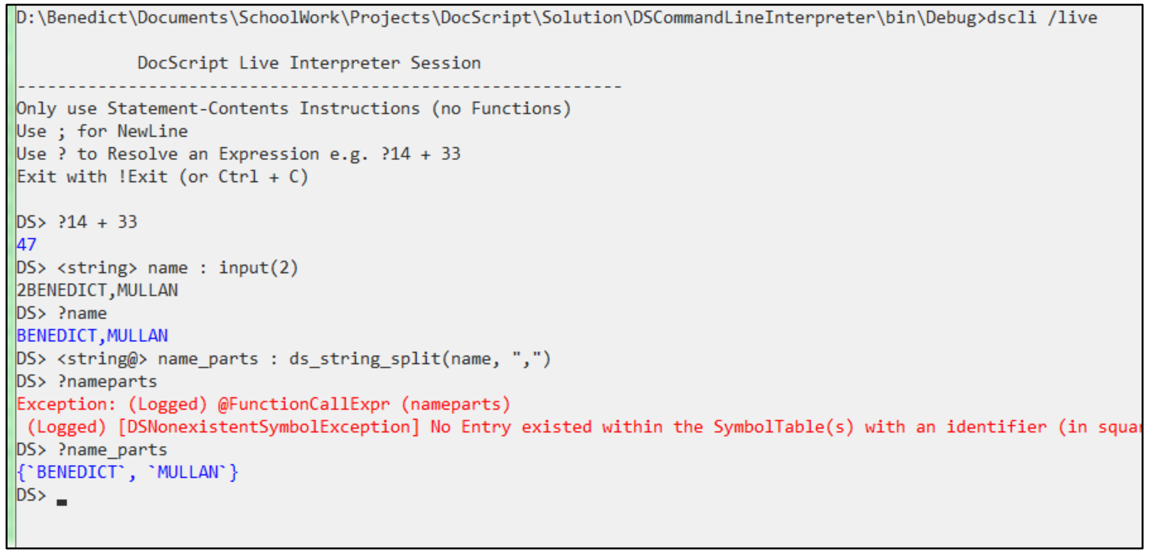
Start
Pictorial Help!

++ DocScript Remoting

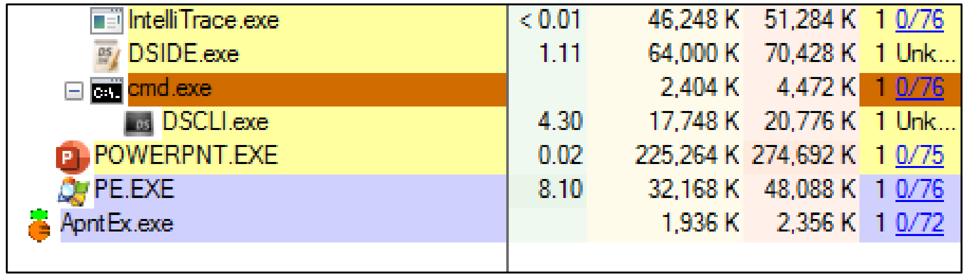
DocScript Command-Line Interpreter

Implementation of DSIE/DLL into a Windows Interpreter Binary

```
Description:
-------------------------------------------------------------------------------
DocScript Command-Line Interpreter. Interprets DocScript Source Files.

Examples:
-------------------------------------------------------------------------------
DSCLI.EXE /Live
DSCLI.EXE /Live /LogToFile:"DSLive.DSLog" /ProcessDebugEvents /GUI
DSCLI.EXE /Run /SourceString:"Function <Void> Main ();Output(`Hello, World!`);EndFunction"
DSCLI.EXE /Run /SourceFile:"X:\Programming\DocScript\HelloWorld.DS" /LogToConsole
DSCLI.EXE /GetProgramTree /SourceString:"Function <Void> Main ();Output(`Hello, World!`);EndFunction"
DSCLI.EXE /Run /SourceString:"Function<Void>Main();System_Beep();EndFunction"
DSCLI.EXE /Run /SourceFile:"BIO2017.DS" /DocScriptCLAs:"GRBBRB" /LogToFile:BIO.DSLog

Argument Usage: (Keys are case-insensitive)
-------------------------------------------------------------------------------
/Live                         (Optional) [Action] Enters a DocScript Live Sessio
                              n: a DS> prompt appears and accepts Statement-leve
                              l Instructions
/Run                          (Optional) [Action] Interprets the DocScript Sourc
                              e (specified by either /SourceFile or /SourceStrin
                              g). This process then returns the ExitCode of the
                              DocScript Program.
/GetProgramTree               (Optional) [Action] Parses and Lexes the DocScript
                               Source (specified by either /SourceFile or /Sourc
                              eString), and writes the resultant XML Program tre
                              e to the Console Output Stream
/SourceFile:<Value>           (Optional) [Datum] Specifies the Source via a DocS
                              cript Source File
/SourceString:<Value>         (Optional) [Datum] Specifies the Source via a DocS
                              cript Source String. Use ; for NewLine and ` for S
                              tringLiteralStartEndChar.
/DocScriptCLAs:<Value>        (Optional) [Datum] Specifies Command-Line Argument
                              s for the DocScript Program
/LogToConsole                 (Optional) [Flag] Writes Events from the DocScript
                               Log to the Console Output Stream during Interpret
                              ation
/LogToFile:<Value>            (Optional) [Flag+Datum] Writes Events from the Doc
                              Script Log to the specified Text File during Inter
                              pretation
/ProcessDebugEvents           (Optional) [Flag] Processes and shows Debugging Me
                              ssages in the Log (if the Log is shown)
/GUI                          (Optional) [Flag] Indicates that the GUI Execution
                              Context will be used instead of the CLI one
```



```
D:\Benedict\Documents\SchoolWork\Projects\DocScript\Solution\DSCommandLineInterpreter\bin\Debug>dscli /live

          DocScript Live Interpreter Session
---------------------------------------------------------------
Only use Statement-Contents Instructions (no Functions)
Use ; for NewLine
Use ? to Resolve an Expression e.g. ?14 + 33
Exit with !Exit (or Ctrl + C)

DS> ?14 + 33
47
DS> <string> name : input(2)
2BENEDICT,MULLAN
DS> ?name
BENEDICT,MULLAN
DS> <string@> name_parts : ds_string_split(name, ",")
DS> ?nameparts
Exception: (Logged) @FunctionCallExpr (nameparts)
 (Logged) [DSNonexistentSymbolException] No Entry existed within the SymbolTable(s) with an identifier (in squa
DS> ?name_parts
{`BENEDICT`, `MULLAN`}
DS> _
```



IntelliTrace.exe	< 0.01	46,248 K	51,284 K	1	0/76
DSIDE.exe	1.11	64,000 K	70,428 K	1	Unk...
cmd.exe		2,404 K	4,472 K	1	0/76
DSCLI.exe	4.30	17,748 K	20,776 K	1	Unk...
POWERPNT.EXE	0.02	225,264 K	274,692 K	1	0/75
PE.EXE	8.10	32,168 K	48,088 K	1	0/76
ApntEx.exe		1,936 K	2,356 K	1	0/72

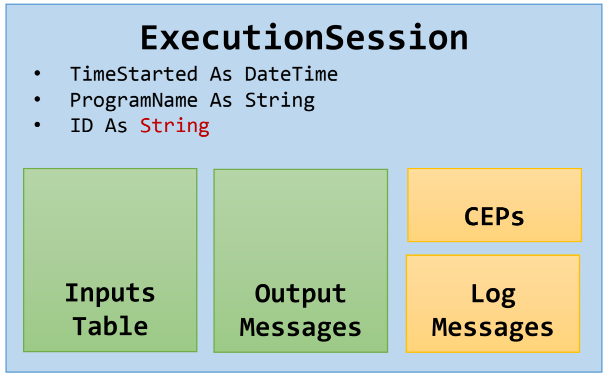
DocScript Interactive [0]

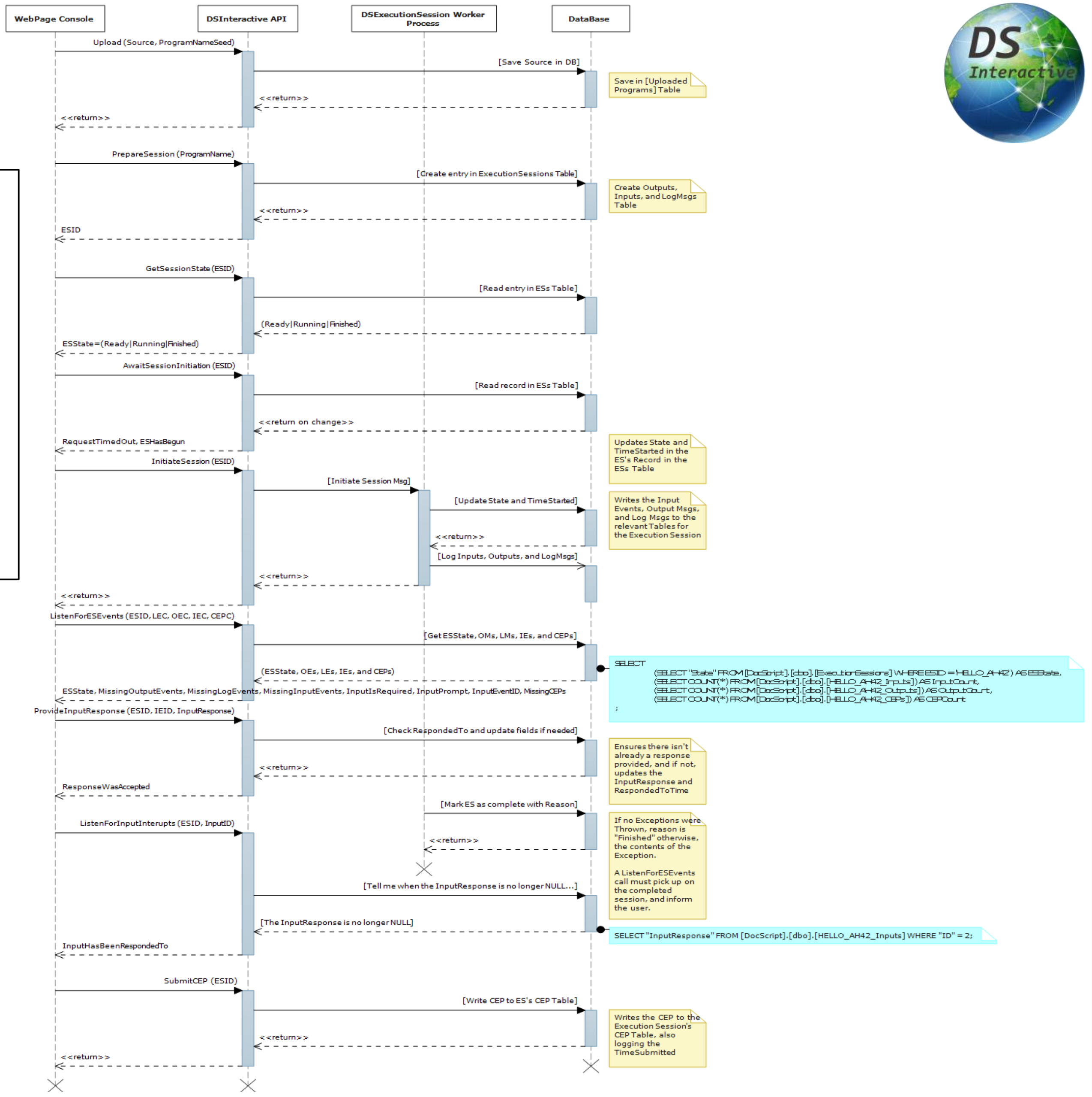
Distributed multi-client browser-based execution and presentation environment

(!Outdated) API EndPoints:

```
-   \
        - ExprTree.ASPX
              - ?Action=GetExprTree&Expr=15^3
-   Interactive\
        - Upload.ASPX
              - ?Item=DSSource&ProgramNameSeed=HelloWorld.DS
        - ExecutionSession.ASPX
              - ?Action=PrepareSession&ProgramName=HelloWorld.DS
              - ?Action=InitiateSession&ESID=HELLO_AH42
              - ?Action=ListenForExecutionEvents&SessionID=HEL001&OutputMsgCount=3&LogMsgC
                ount=14
              - ?Action=ProvideInputResponse&InputEventID=1&InputResponse=17
              - ?Action=ListenForInputRequestInterupts&InputEventID=1
              - ?Action=GetExecutionSessionState&SessionID=HELLO_AH46
```

++ Installable DSI PWA (Progressive Web Application)




WebPage Console
DSInteractive API
DSExecutionSession Worker Process
DataBase
Upload (Source, ProgramNameSeed)
[Save Source in DB]
Save in [Uploaded Programs] Table
<<return>>
<<return>>
PrepareSession (ProgramName)
[Create entry in ExecutionSessions Table]
Create Outputs, Inputs, and LogMsgs Table
<<return>>
ESID
GetSessionState (ESID)
[Read entry in ESs Table]
(Ready|Running|Finished)
ESState=(Ready|Running|Finished)
AwaitSessionInitiation (ESID)
[Read record in ESs Table]
<<return on change>>
RequestTimedOut, ESHasBegun
Updates State and TimeStarted in the ES's Record in the ESs Table
InitiateSession (ESID)
[Initiate Session Msg]
[Update State and TimeStarted]
Writes the Input Events, Output Msgs, and Log Msgs to the relevant Tables for the Execution Session
<<return>>
[Log Inputs, Outputs, and LogMsgs]
<<return>>
<<return>>
ListenForESEvents (ESID, LEC, OEC, IEC, CEPC)
[Get ESState, OMs, LMs, IEs, and CEPs]
(ESState, OEs, LEs, IEs, and CEPs)
ESState, MissingOutputEvents, MissingLogEvents, MissingInputEvents, InputIsRequired, InputPrompt, InputEventID, MissingCEPs
ProvideInputResponse (ESID, IEID, InputResponse)
[Check RespondedTo and update fields if needed]
Ensures there isn't already a response provided, and if not, updates the InputResponse and RespondedToTime
<<return>>
ResponseWasAccepted
[Mark ES as complete with Reason]
If no Exceptions were Thrown, reason is "Finished" otherwise, the contents of the Exception.
A ListenForESEvents call must pick up on the completed session, and inform the user.
<<return>>
ListenForInputInterupts (ESID, InputID)
[Tell me when the InputResponse is no longer NULL...]
[The InputResponse is no longer NULL]
InputHasBeenRespondedTo
SubmitCEP (ESID)
[Write CEP to ES's CEP Table]
Writes the CEP to the Execution Session's CEP Table, also logging the TimeSubmitted
<<return>>
<<return>>

```
SELECT
        (SELECT "State" FROM [DocScript].[dbo].[ExecutionSessions] WHERE ESID = 'HELLO_AH42') AS ESState,
        (SELECT COUNT(*) FROM [DocScript].[dbo].[HELLO_AH42_Inputs]) AS InputCount,
        (SELECT COUNT(*) FROM [DocScript].[dbo].[HELLO_AH42_Outputs]) AS OutputCount,
        (SELECT COUNT(*) FROM [DocScript].[dbo].[HELLO_AH42_CEPs]) AS CEPCount
;
```

```
SELECT "InputResponse" FROM [DocScript].[dbo].[HELLO_AH42_Inputs] WHERE "ID" = 2;
```

Distinguish WasLegitimateRequest and OperationCompletedSuccessfully

?Action=ListenForExecutionEvents&SessionID=HEL001&OutputMsgCount=3&LogMsgCount=14
Returns: MissingOutputMsgs; MissingLogMsgs; InputIsRequired; InputPrompt; InputEventID

ESParticipant.ASPX?SessionID=HEL001&Role=Observer&Name=Ben

DSIExecutionSession.exe /ESID:HELLO_AH42

/API/Interactive/?Action=AwaitSessionInitiation&ESID=HELLO_AH42

DocScript Interactive [1]

Distri[illegible] multi-client browser-based execution and presentation environment

IDs start at 1. This is so that 0 can be used to mean [none of the records].

"

(!Outdated)

↓ ExecutionSes[illegible]n Client Execution Packages (CEPs) Table ↓

ID	Tim[illegible]omitted	JavaScriptCEP
1	1:32:49 [illegible]08-2022	Fun[illegible]() { … }
2	1:32:49 12-08-2022	Func[illegible]() { … }
3	1:32[illegible]12-08-202[illegible]	Fun[illegible]n () {[illegible]
4	1:32:49 12-08-2022	F[illegible]on ([illegible] }
5	1:32:49 12-08-2022	[illegible]unction[illegible].. }

↓ Uploaded Programs Table ↓

ID	TimeSubmitted	Source	ProgramName
1	1:32:49 12-08-2022	Function <Void> Main () …	HelloWorld
2	1:32:49 12-08-2022	Function <Void> Main () …	InputName
3	1:32:49 12-08-2022	Function <Void> Main () …	AgeTest
4	1:32:49 12-08-2022	Function <Void> Main () …	Program1
5	1:32:49 12-08-2022	Function <Void> Main () …	PROGRAM3

↓ ExecutionSe[illegible] Table ↓

ESID	ProgramName	TimeStarted	TimeFinished	[illegible]ate	ExitRe[illegible]
HELLO_AH42	HelloWorld.DS			Rea[illegible]	
HELLO_AH43	HelloWorld.DS	1:32:49 12-08-2022		Running	
HELLO_AH44	HelloWorld.DS	1:32:49 12-08-2022	1:32:49 12-08-2022	Finished	Finished Successfully
HELLO_AH45	HelloWorld.DS	1:32:49 12-08-2022	1:32:49 12-08-2022	Finished	Input Timed Out for "…"
HELLO_AH46	HelloWorld.DS	1:32:49 12-08-2022	1:32:49 12-08-2022	Finished	Finished Successfully

↓ ExecutionSession Outputs Table ↓

ID	TimeSubmitted	OutputMessage
1	1:32:49 12-08-2022	"Enter Name"
2	1:32:49 12-08-2022	"Enter Age"
[illegible]	1:32:49 12-08-2022	"Enter A"
4	1[illegible]-2022	"Enter[illegible]"
5	[illegible]12-[illegible]022	"Ente[illegible]"

↓ ExecutionSession Inputs Table ↓

ID	TimeSubmitted	InputPrompt	InputResponse	RespondedTo	RespondedToTime
1	1:32:49 12-08-2022	"Enter Name"	"Ben"	True	1:32:49 12-08-2022
2	1:32:49 12-08-2022	"Enter Age"		False	1:32:49 12-08-2022
3	1:32:49 12-08-2022	"Enter A"		False	1:32:49 12-08-2022
4	1:32:49 12-08-2022	"Enter B"		False	1:32:49 12-08-2022
5	1:32:49 12-08-2022	"Enter C"		False	1:32:49 12-08-2022

↓ ExecutionS[illegible]ion LogM[illegible] Table

ID	TimeSubmitted	LogMessage	[illegible]verit[illegible]	Catago[illegible]
1	1:32:49 12-08-2022	"Enter Name"	Error	Execution
2	1:32:49 12-08-2022	"Enter Age"	Warning	Lexing
3	1:32:49 12-08-2022	"Enter A"	Information	[illegible]
4	1:32:49 12-08-2022	"Enter B"	Verbose	Unspecified
5	1:32:49 12-08-2022	"Enter C"	Debug	System

DSI [2] Database Tables

IDs start at 1. This is so that 0 can be used to mean [none of the records].

```
SELECT
                        (SELECT "State" FROM [DocScript].[dbo].[ExecutionSessions] WHERE ESID = 'HELLO_AH42') AS ESState,
                        (SELECT COUNT(*) FROM [DocScript].[dbo].[HELLO_AH42_Inputs]) AS InputCount,
                        (SELECT COUNT(*) FROM [DocScript].[dbo].[HELLO_AH42_Outputs]) AS OutputCount,
                        (SELECT COUNT(*) FROM [DocScript].[dbo].[HELLO_AH42_CEPs]) AS CEPCount
;
```

↓ ExecutionSessions Table ↓

ESID	ProgramName	TimeStarted	TimeEnded	State	ExitReason
HELLO_AH42	HelloWorld.DS	NULL	NULL	Ready	NULL
HELLO_AH43	HelloWorld.DS	1:32:49 12-08-2022	NULL	Running	NULL
HELLO_AH44	HelloWorld.DS	1:32:49 12-08-2022	1:32:49 12-08-2022	Ended	Normal DSExitCode=5
HELLO_AH45	HelloWorld.DS	1:32:49 12-08-2022	1:32:49 12-08-2022	Ended	Input Timed Out for "…"
HELLO_AH46	HelloWorld.DS	1:32:49 12-08-2022	1:32:49 12-08-2022	Ended	Normal DSExitCode=0

↓ UploadedPrograms Table ↓

TimeSubmitted	Source	ProgramName
1:32:49 12-08-2022	Function <Void> Main () …	HelloWorld.DS
1:32:49 12-08-2022	Function <Void> Main () …	InputName
1:32:49 12-08-2022	Function <Void> Main () …	AgeTest
1:32:49 12-08-2022	Function <Void> Main () …	Program1
1:32:49 12-08-2022	Function <Void> Main () …	PROGRAM3

↓ ExecutionSession Inputs Table ↓

ID	TimeSubmitted	InputPrompt	InputResponse	RespondedToTime
1	1:32:49 12-08-2022	"Enter Name"	"Ben"	1:32:49 12-08-2022
2	1:32:49 12-08-2022	"Enter Age"	NULL	NULL
3	1:32:49 12-08-2022	"Enter A"	NULL	NULL
4	1:32:49 12-08-2022	"Enter B"	NULL	NULL
5	1:32:49 12-08-2022	"Enter C"	NULL	NULL

↓ ExecutionSession Outputs Table ↓

ID	TimeSubmitted	OutputMessage
1	1:32:49 12-08-2022	"Hello, World!"
2	1:32:49 12-08-2022	"Enter Age"
3	1:32:49 12-08-2022	"Enter A"
4	1:32:49 12-08-2022	"Enter B"
5	1:32:49 12-08-2022	"Enter C"

API EndPoint "*ListenForESEvents*"

d

```
SELECT "InputResponse" FROM [DocScript].[dbo].[HELLO_AH42_Inputs] WHERE "ID" = 2;
```

↓ ExecutionSession Client Execution Packages (CEPs) Table ↓

ID	TimeSubmitted	JavaScriptToRun
1	1:32:49 12-08-2022	Function () { … }
2	1:32:49 12-08-2022	Function () { … }
3	1:32:49 12-08-2022	Function () { … }
4	1:32:49 12-08-2022	Function () { … }
5	1:32:49 12-08-2022	Function () { … }

↓ ExecutionSession LogEvents Table ↓

ID	TimeSubmitted	Message	Severity	Category
1	1:32:49 12-08-2022	"Enter Name"	Error	Execution
2	1:32:49 12-08-2022	"Enter Age"	Warning	Lexing
3	1:32:49 12-08-2022	"Enter A"	Information	Parsing
4	1:32:49 12-08-2022	"Enter B"	Verbose	Unspecified
5	1:32:49 12-08-2022	"Enter C"	Debug	System

DSI [3] Database Validation & Size Limits

varchar ONLY HOLDS ASCII!!!

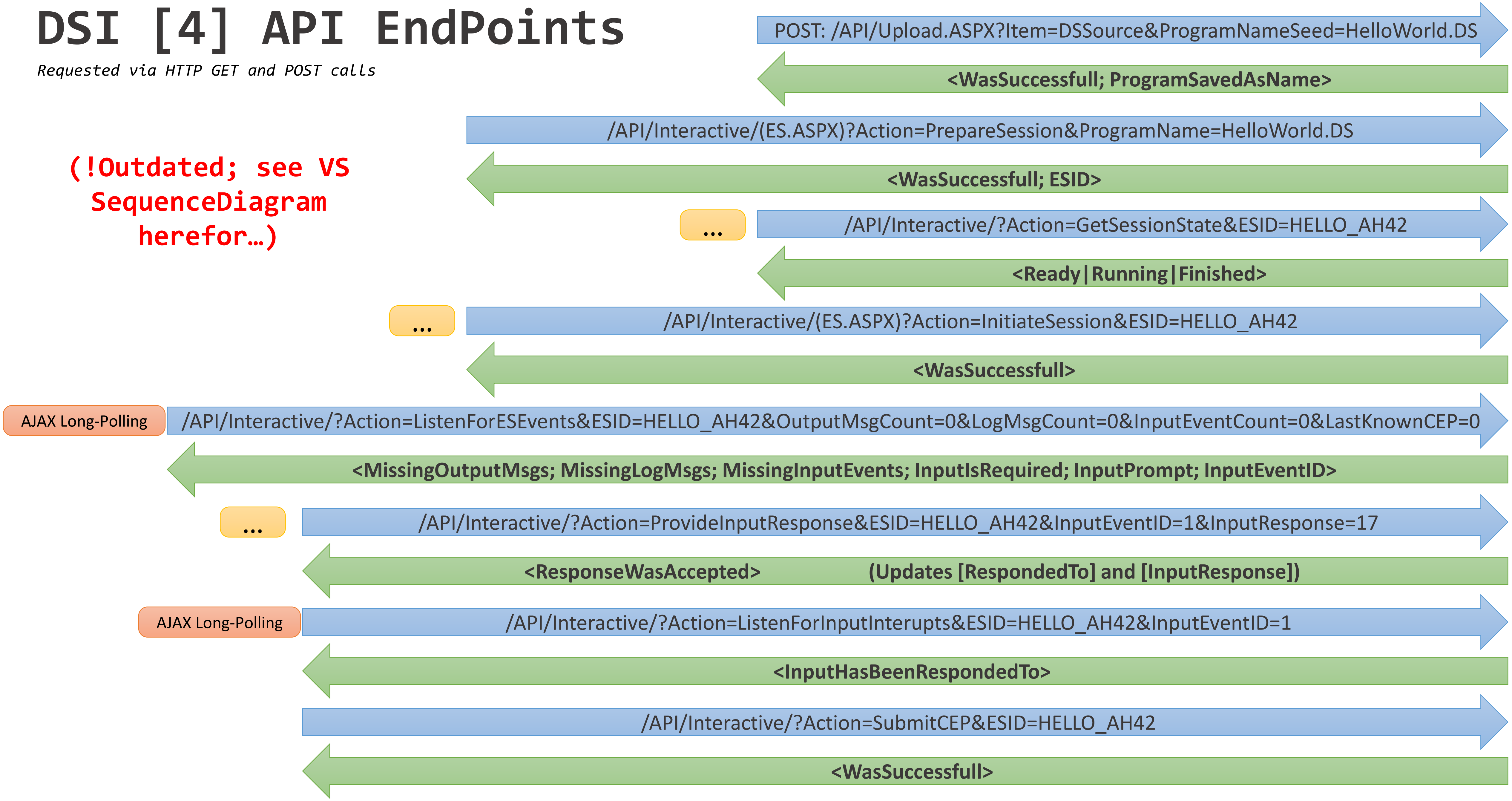
- UploadedPrograms\ProgramName is varchar(100)
- ExecutionSessions\ESID is varchar(100)
- ExecutionSessions\State is **varchar(100)** **For consistency…**
- ExecutionSessions\ProgramName is varchar(100)
- {ESID}_LogEvents\Severity is varchar(100)
- {ESID}_LogEvents\Category is varchar(100)

!DocScript: LogEvents, InputEvents, OutputEvents, and CEPs cannot be removed or modified after they have been added

DSI [4] API EndPoints

Requested via HTTP GET and POST calls

(!Outdated; see VS SequenceDiagram herefor…)



POST: /API/Upload.ASPX?Item=DSSource&ProgramNameSeed=HelloWorld.DS

<WasSuccessfull; ProgramSavedAsName>

/API/Interactive/(ES.ASPX)?Action=PrepareSession&ProgramName=HelloWorld.DS

<WasSuccessfull; ESID>

…

/API/Interactive/?Action=GetSessionState&ESID=HELLO_AH42

<Ready|Running|Finished>

…

/API/Interactive/(ES.ASPX)?Action=InitiateSession&ESID=HELLO_AH42

<WasSuccessfull>

AJAX Long-Polling

/API/Interactive/?Action=ListenForESEvents&ESID=HELLO_AH42&OutputMsgCount=0&LogMsgCount=0&InputEventCount=0&LastKnownCEP=0

<MissingOutputMsgs; MissingLogMsgs; MissingInputEvents; InputIsRequired; InputPrompt; InputEventID>

…

/API/Interactive/?Action=ProvideInputResponse&ESID=HELLO_AH42&InputEventID=1&InputResponse=17

<ResponseWasAccepted> **(Updates [RespondedTo] and [InputResponse])**

AJAX Long-Polling

/API/Interactive/?Action=ListenForInputInterupts&ESID=HELLO_AH42&InputEventID=1

<InputHasBeenRespondedTo>

/API/Interactive/?Action=SubmitCEP&ESID=HELLO_AH42

<WasSuccessfull>

DSI [5] Client Pages

HTML for the Client's Browser

?Role=Observer *(No CEP-Insertion or Input-Responding)*
?Role=Responder *(No CEP-Insertion, but Input-Responding)*
?Role=Controller *(CEP-Insertion and Input-Responding)*

Required functions:
- *Create ES*
- *Participate in ES*
- *View old ESs*

Pages:
- ESParticipant.ASPX?ESID=*&Role=*
- ESManager.ASPX
- UploadProgram.ASPX

Interactive/ESParticipant

Outputs
[12:14:00] Hello, World!
[12:14:01] Hello, World!
[12:14:02] Hello, World!

Inputs
[14:15:00] (Enter your name) "Ben Mullan"
[14:15:00] (Enter your name) "Ben Mullan"
[14:15:00] (Enter your name) "Ben Mullan"

Log Events:
0 executed

CEPs: 0 executed

CEPs: 0 executed

Interactive/ESManager

Execution Sessions
HELLO_AH42 – *HelloWorld.DS*: Ready
HELLO_AH42 – *HelloWorld.DS*: Ready
HELLO_AH42 – *HelloWorld.DS*: Running (started 08:15)
HELLO_AH42 – *HelloWorld.DS*: Ended

[+ Add]
Can Prepare an ES from an UploadedProgram,
Or immediately Prepare&Initiate an ES from an UploadedProgram

UploadProgram

Source
Function <Void> Main () …

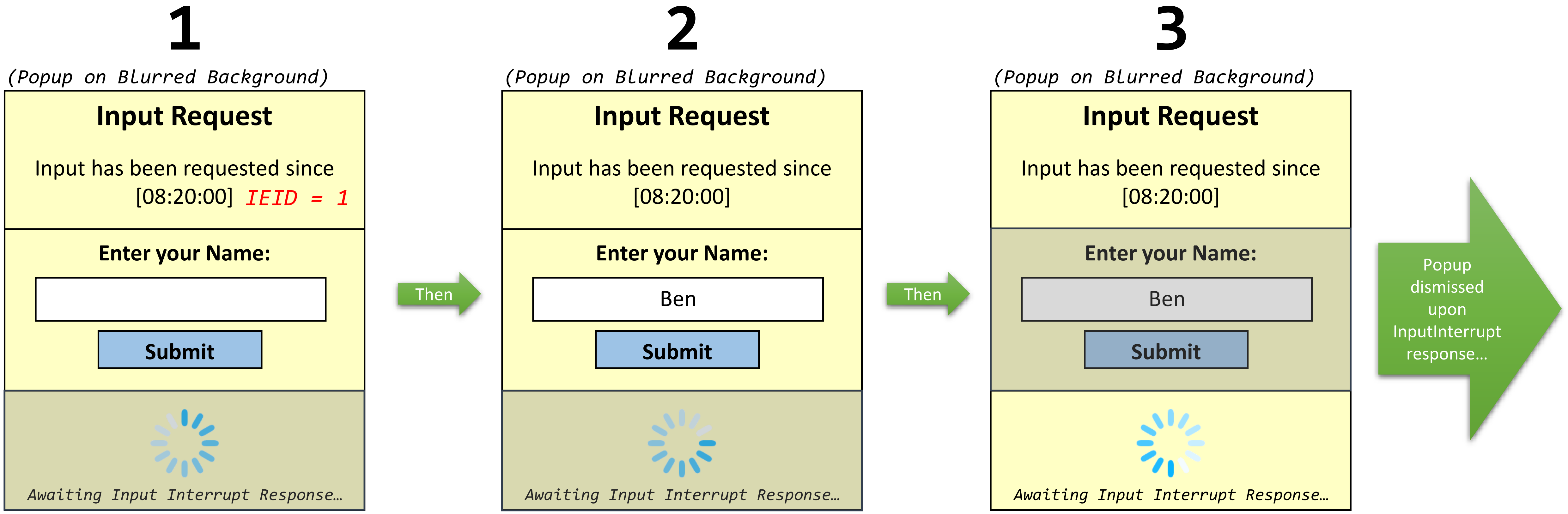
[Upload]

[Run]
(Prepares a New ES, and Initiates it)

DSI [6] Client Input Procedure [0] *(All on ESParticipant.ASPX)*

Input procedure 0 – the client responds to the InputRequest…

Remember to "ALTER DATABASE [DocScript] SET ENABLE_BROKER;" in order for [SqlDependancy]s to work...


1
(Popup on Blurred Background)
Input Request
Input has been requested since [08:20:00] IEID = 1
Enter your Name:
Submit
Awaiting Input Interrupt Response…
Then
2
(Popup on Blurred Background)
Input Request
Input has been requested since [08:20:00]
Enter your Name:
Ben
Submit
Awaiting Input Interrupt Response…
Then
3
(Popup on Blurred Background)
Input Request
Input has been requested since [08:20:00]
Enter your Name:
Ben
Submit
Awaiting Input Interrupt Response…
Popup dismissed upon InputInterrupt response…

[InputIsRequired] comes back as True; the Client sends a ListenForInputInterupts request…

Client enters InputResponse and clicks Submit

Client must await InputInterrupt Response…

? OperationWasSuccessfull vs. UnhandledExceptionOccured

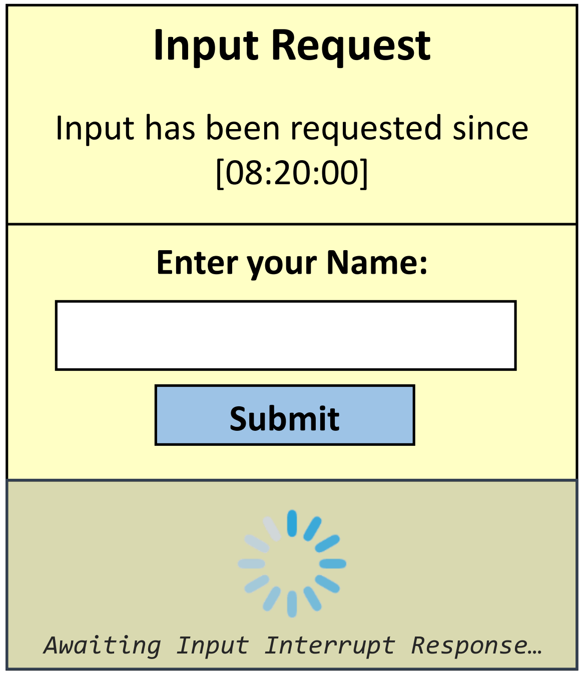
!Maybe say "Listening for external Input Responses…" instead of "Awaiting Input Interrupt Response…"

DSI [7] Client Input Procedure [1] *(All on ESParticipant.ASPX)*

Input procedure 0 – another client responds to the InputRequest first…

1

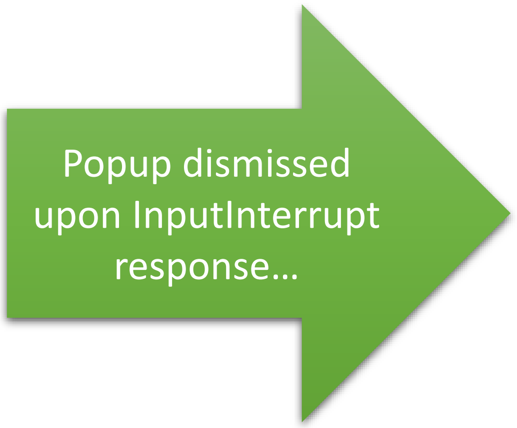
(Popup on Blurred Background)



Input Request

Input has been requested since [08:20:00]

Enter your Name:

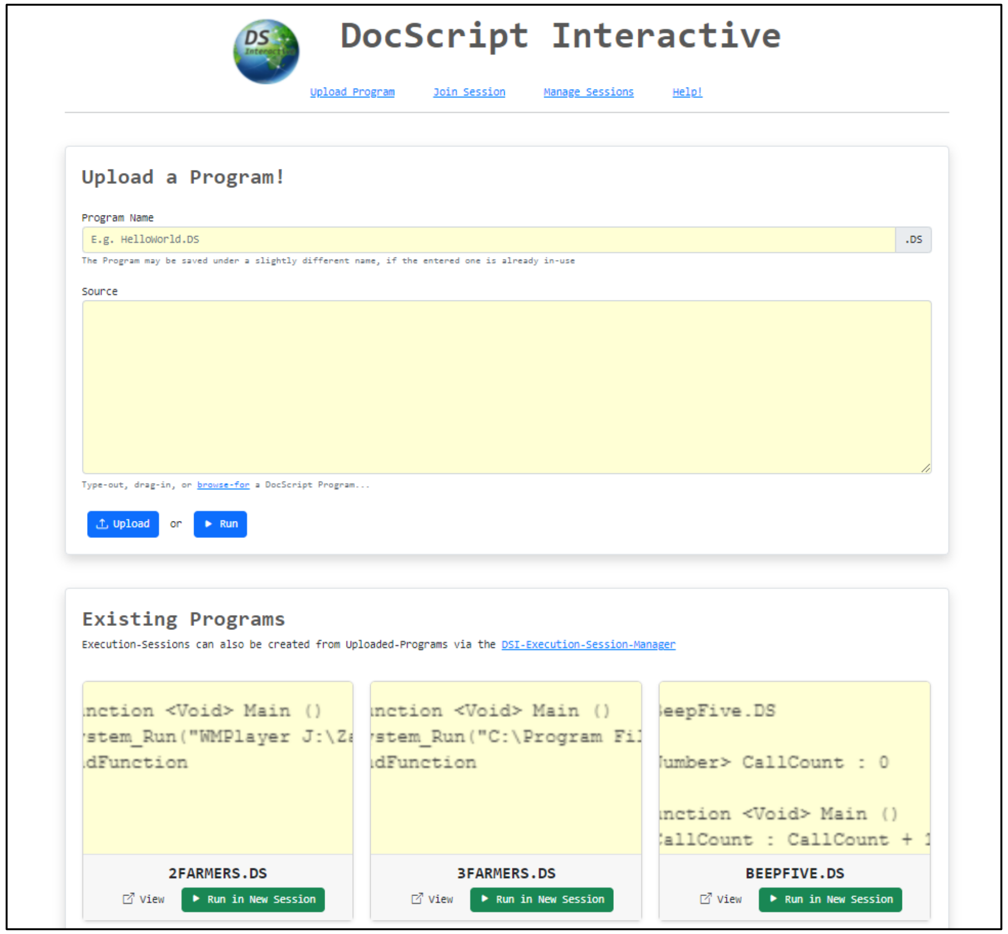


[InputIsRequired] comes back as True; the Client sends a ListenForInputInterupts request…

Another client provided an InputResponse first, so the dialog is closed and execution continues…

DSI [8] Final Product

How it actually came out…



DocScript Interactive

Upload Program   Join Session   Manage Sessions   Help!

Upload a Program!

Program Name

E.g. HelloWorld.DS .DS

The Program may be saved under a slightly different name, if the entered one is already in-use

Source

Type-out, drag-in, or browse-for a DocScript Program...

Upload or Run

Existing Programs

Execution-Sessions can also be created from Uploaded-Programs via the DSI-Execution-Session-Manager

```
nction <Void> Main ()
stem_Run("WMPlayer J:\Z
dFunction
```

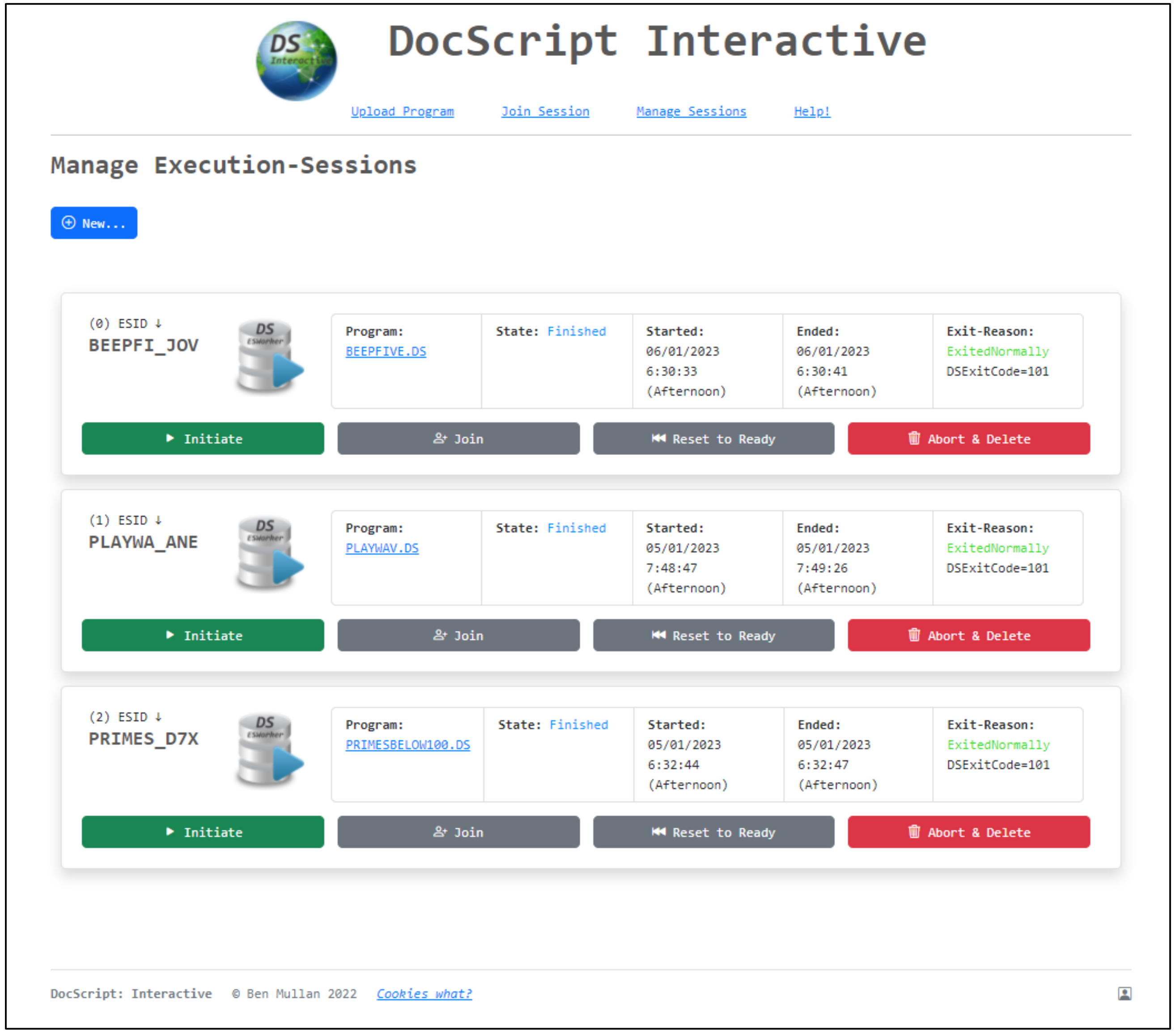
2FARMERS.DS — View | Run in New Session

```
nction <Void> Main ()
stem_Run("C:\Program Fi
dFunction
```

3FARMERS.DS — View | Run in New Session

```
eepFive.DS

umber> CallCount : 0

nction <Void> Main ()
allCount : CallCount + 1
```

BEEPFIVE.DS — View | Run in New Session



DocScript Interactive

Upload Program   Join Session   Manage Sessions   Help!

Manage Execution-Sessions

New...

ESID	Program	State	Started	Ended	Exit-Reason
(0) ESID ↓ BEEPFI_JOV	BEEPFIVE.DS	Finished	06/01/2023 6:30:33 (Afternoon)	06/01/2023 6:30:41 (Afternoon)	ExitedNormally DSExitCode=101
(1) ESID ↓ PLAYWA_ANE	PLAYWAV.DS	Finished	05/01/2023 7:48:47 (Afternoon)	05/01/2023 7:49:26 (Afternoon)	ExitedNormally DSExitCode=101
(2) ESID ↓ PRIMES_D7X	PRIMESBELOW100.DS	Finished	05/01/2023 6:32:44 (Afternoon)	05/01/2023 6:32:47 (Afternoon)	ExitedNormally DSExitCode=101

Initiate | Join | Reset to Ready | Abort & Delete

DocScript: Interactive © Ben Mullan 2022 *Cookies what?*

DSI [9] Mass-Testing

Distributed, Multi-Client Testing

Binaries

Compiled Binaries for the DocScript Logic and Implementations

File Name	Description
DocScript.Library.DLL	Core Interpretation Logic and Utilities DLL
DSIDE.EXE	DocScript Windows IDE Executable
DSCLI.EXE	DocScript Command-Line Interpreter Executable
DSIExecutionSessionWorker.EXE	Worker Executable for DocScript Interactive Sessions
DSExpr.EXE	Standalone Expression Resolution Executable
DocScript.WebParts.DLL	DocScript WebParts Resource DLL

(More) Examples

Samples of DocScript Source

```
Function <Void> Main ()
    Output("Hello, World!")
    Return
EndFunction
```

```
Function <Number> Main (<String@> _CLAs)
    Output("First CLA: " & StringArray_At(_CLAs, 0))
    Return 0
EndFunction
```

```
DSFunction
            @Name         = "Main"
            @ReturnType   = DSNumber
            @Arguments    = {<String@> "_CLAs"}
            FunctionCall
                        @FunctionName= "Output"
                        @Arguments   = {<String>}
            ReturnToCaller
                        @ReturnValue = <IExpression>
```

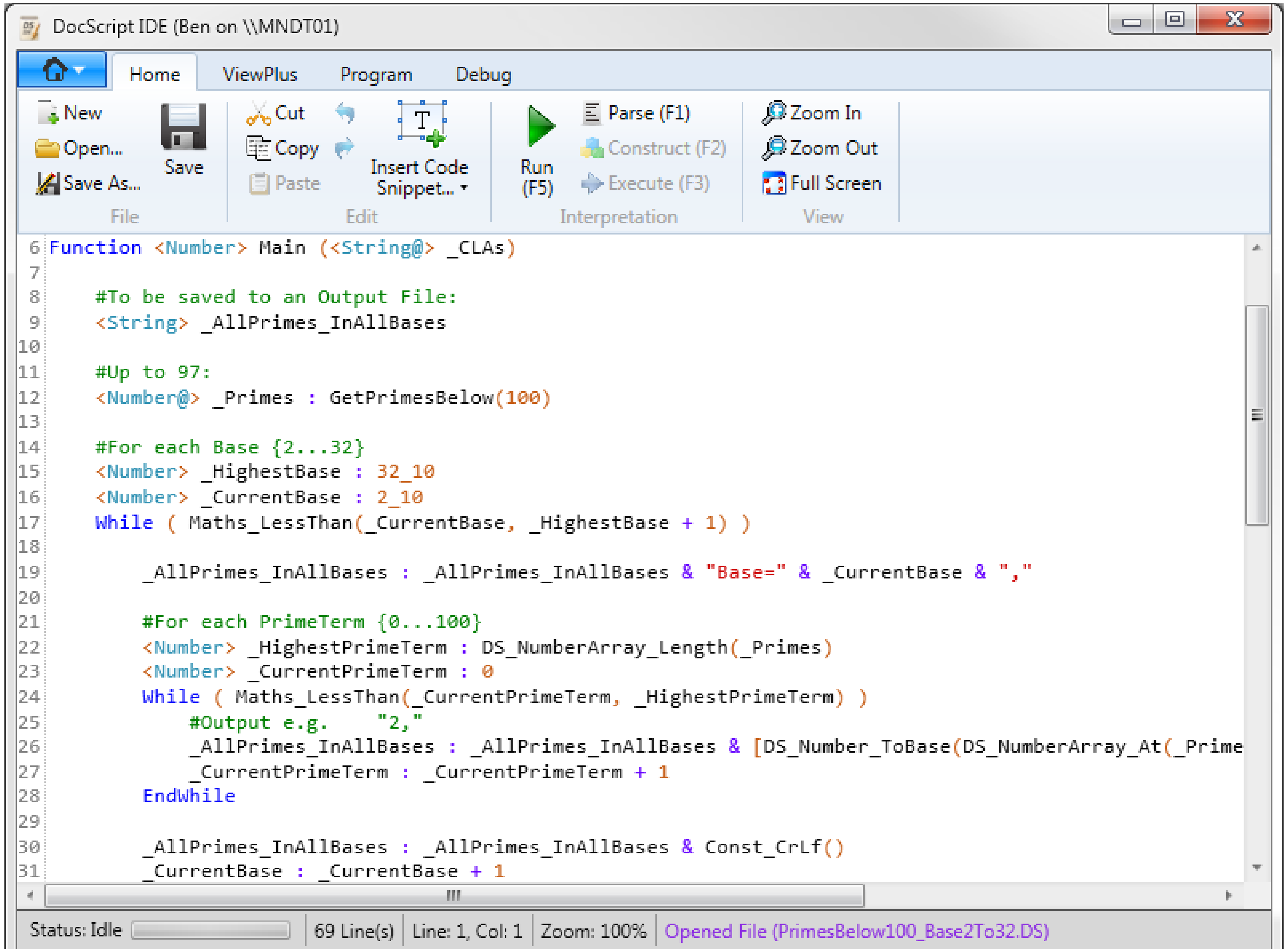
Decision: No $ in v1

```
Function <Number> Main (<String@> _CLAs)

    #Input looks like "-Name" "Ben" "-Age" "13"
    #Get the Value for an Input()'ed Key

    <String> _Key = Input("Argument Key:")
    <String> _Value = GetCLAValueFromKey(_Key)

    Output("Value: " & _Value)
    Return 0

EndFunction

Function <String> GetCLAValueFromKey(<String@> _CLAs, <String> _Key)

    Loop (StringArray_MaxIndex(_CLAs))
        If (StringArray_At(_CLAs, $) == ("-" & _Key))
            Return StringArray_At(_CLAs, $ + 1)
        EndIf
    EndLoop

    Return "No Value found for Key [" & _Key & "]"

EndFunction
```

```
Function <Number> Main (<String@> _CLAs)

    #Input looks like "-Name" "Ben" "-Age" "13"
    #Get the Value for an Input()'ed Key

    <String> _Key : Input("Argument Key:")
    <String> _Value : GetCLAValueFromKey(_Key)

    Output("Value: " & _Value)
    Return 0

EndFunction

Function <String> GetCLAValueFromKey(<String@> _CLAs, <String> _Key)

    <Number> _CurrentCLAIndex : 0

    While (_CurrentCLAIndex < [Array_MaxIndex(_CLAs) + 1])
        If (Array_At(_CLAs, _CurrentCLAIndex) = ["-" & _Key])
            Return StringArray_At(_CLAs, _CurrentCLAIndex + 1)
        EndIf
        CurrentCLAIndex : [CurrentCLAIndex + 1]
    EndWhile

    Return "No Value found for Key [" & _Key & "]"

EndFunction
```

(Even More) Examples

Samples of DocScript Source

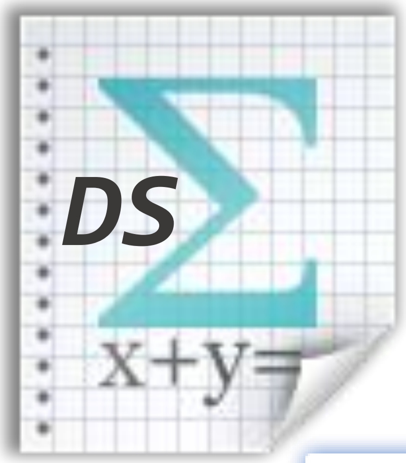


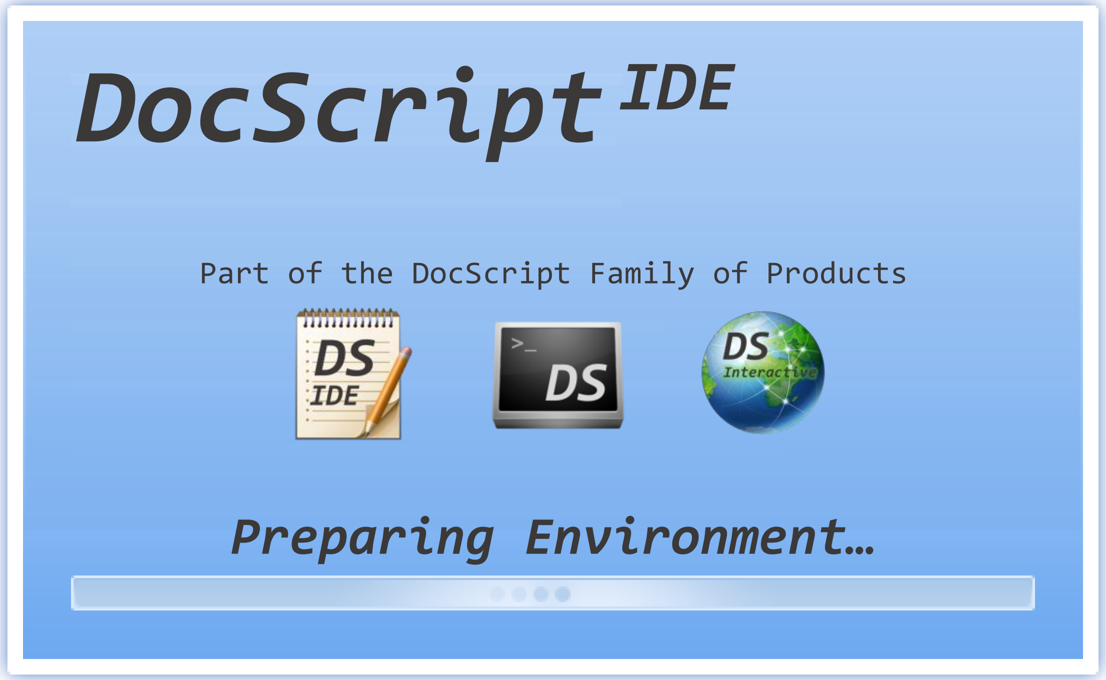
```
Function <Number> Main (<String@> _CLAs)

    #To be saved to an Output File:
    <String> _AllPrimes_InAllBases

    #Up to 97:
    <Number@> _Primes : GetPrimesBelow(100)

    #For each Base {2...32}
    <Number> _HighestBase : 32_10
    <Number> _CurrentBase : 2_10
    While ( Maths_LessThan(_CurrentBase, _HighestBase + 1) )

        _AllPrimes_InAllBases : _AllPrimes_InAllBases & "Base=" & _CurrentBase & ","

        #For each PrimeTerm {0...100}
        <Number> _HighestPrimeTerm : DS_NumberArray_Length(_Primes)
        <Number> _CurrentPrimeTerm : 0
        While ( Maths_LessThan(_CurrentPrimeTerm, _HighestPrimeTerm) )
            #Output e.g.    "2,"
            _AllPrimes_InAllBases : _AllPrimes_InAllBases & [DS_Number_ToBase(DS_NumberArray_At(_Prime
            _CurrentPrimeTerm : _CurrentPrimeTerm + 1
        EndWhile

        _AllPrimes_InAllBases : _AllPrimes_InAllBases & Const_CrLf()
        _CurrentBase : _CurrentBase + 1
```

Status: Idle | 69 Line(s) | Line: 1, Col: 1 | Zoom: 100% | Opened File (PrimesBelow100_Base2To32.DS)

Icons

Insignia for DocScript Implementations and Documentation





DocScript IDE

Part of the DocScript Family of Products

Preparing Environment…

DocScript

Scripting Language System

Video [0]

Explanation video for DocScript

Use this Audio:

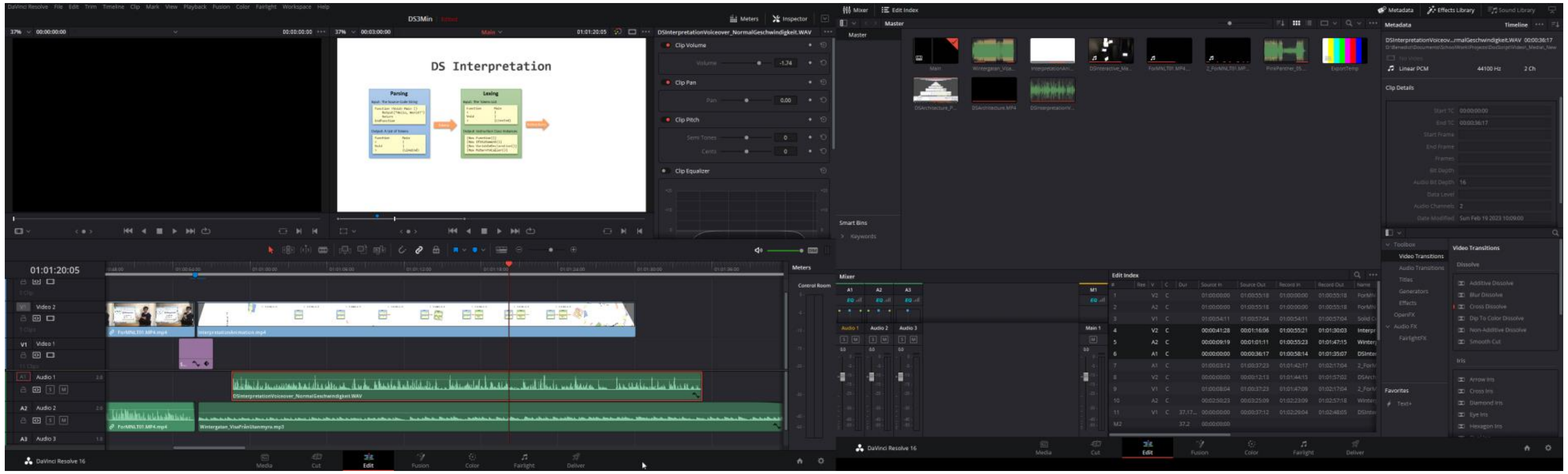
"DocScript in 3 Minutes"

- Architecture
  - Core Interpreter DLL
    - 3 Interpretation Stages
      - Parsing: Token-Types
      - Lexing: 8 IInstruction-Types
      - Execution: Symbol Tables
  - 3 Implementations
    - DSCLI
      - Live
    - DSIDE
      - DSExpr
    - DSInteractive
      - CEP Demonstration (\w 2 Lattitudes)

DocScript is a **simple** procedural programming-language.
So simple, in fact, that there are only 6 keywords, 3 data-types, and HelloWorld looks like this {}
You probably already get the gist of the syntax.

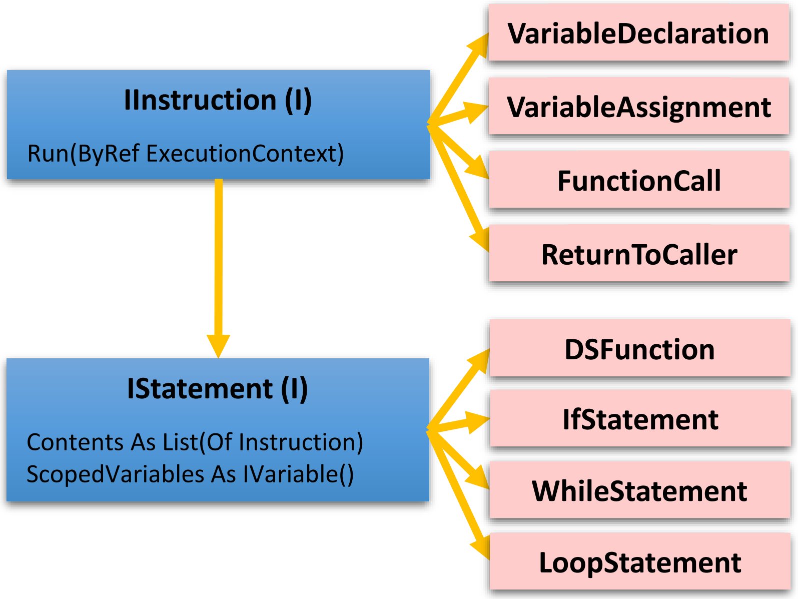
What's different about DocScript, however, is:

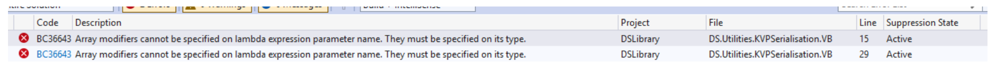
- Native support for numeric literals in different bases (from 2 to 62)
- its ability to support real-time, **multi-client** execution sessions,
- Its highly verbose and detailed output. In fact, the language was designed to be something of a teaching tool; *"Doc-"* comes from the Latin meaning to teach, as in in**doc**trinate or **doc**umentary. If – for instance - you have a malformed expression, DocScript won't just report that there's a syntax-error, but rather, will tell you precisely what's wrong; two binary operators being next to each other for example.



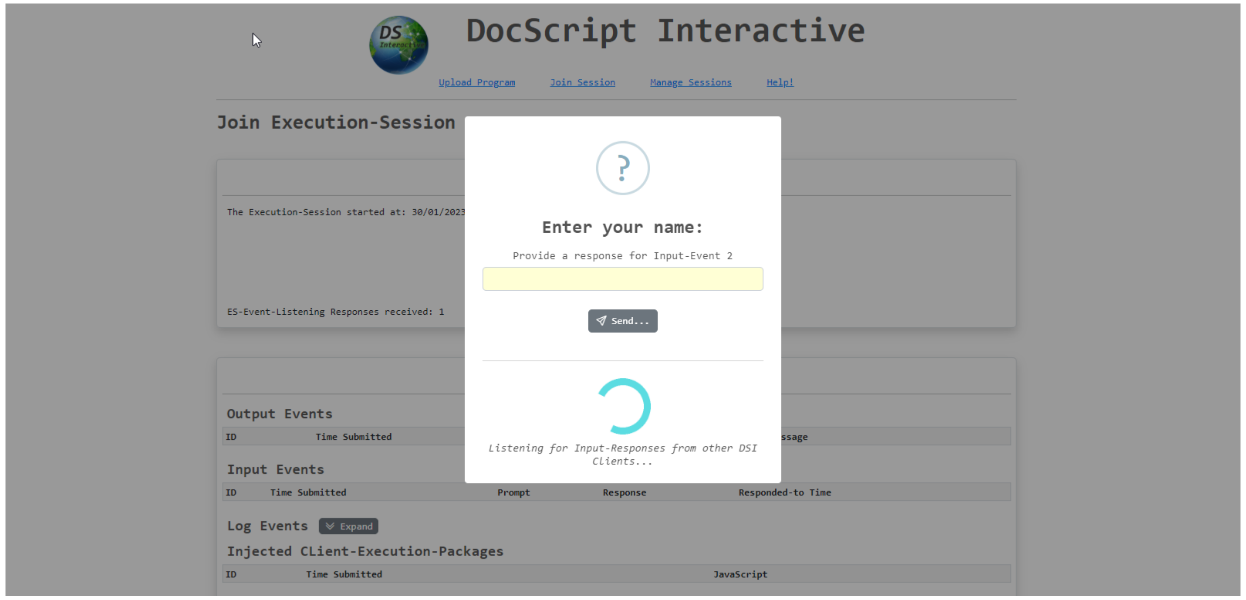
Video [1] Resources

Resources for DS3Min





	Code	Description	Project	File	Line	Suppression State
⊗	BC36643	Array modifiers cannot be specified on lambda expression parameter name. They must be specified on its type.	DSLibrary	DS.Utilities.KVPSerialisation.VB	15	Active
⊗	BC36643	Array modifiers cannot be specified on lambda expression parameter name. They must be specified on its type.	DSLibrary	DS.Utilities.KVPSerialisation.VB	29	Active



DocScript Interactive

Upload Program Join Session Manage Sessions Help!

Join Execution-Session

The Execution-Session started at: 30/01/2023

ES-Event-Listening Responses received: 1

Enter your name:

Provide a response for Input-Event 2

Send...

Listening for Input-Responses from other DSI Clients...

Output Events

ID	Time Submitted	Message

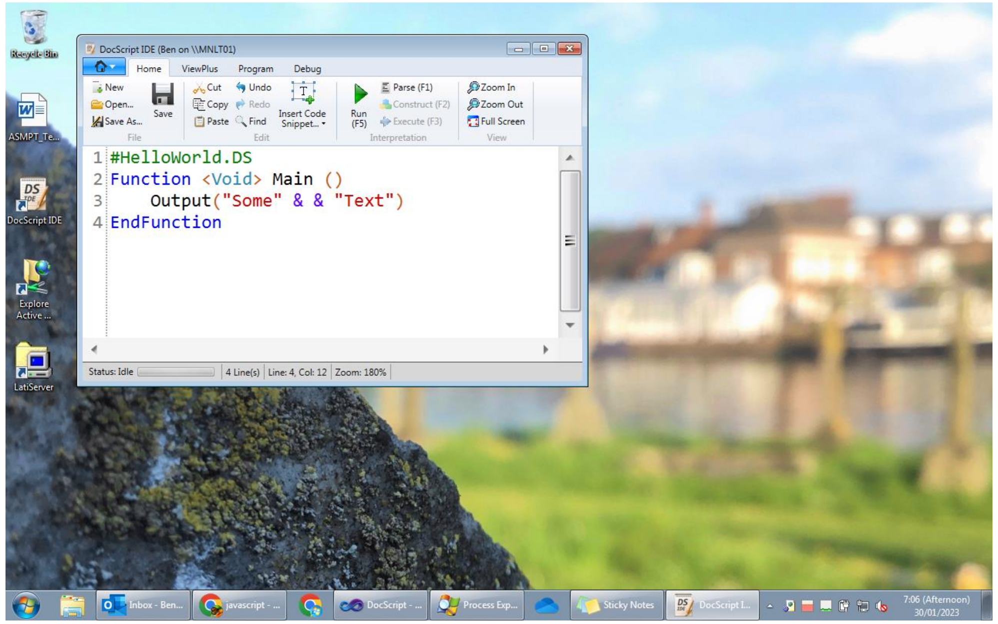
Input Events

ID	Time Submitted	Prompt	Response	Responded-to Time

Log Events Expand

Injected CLient-Execution-Packages

ID	Time Submitted	JavaScript

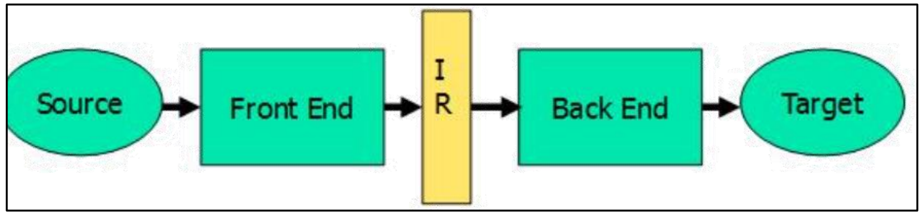


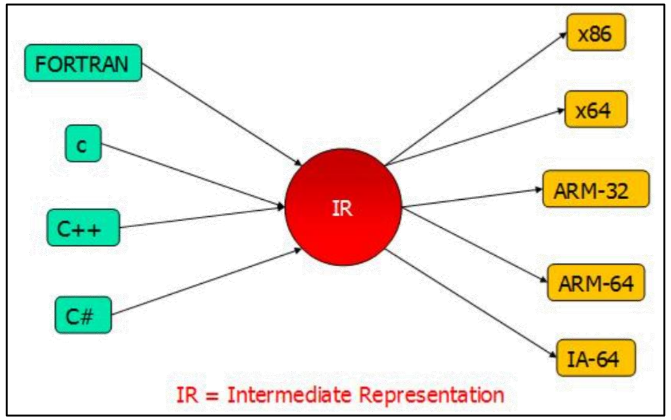
```
#HelloWorld.DS
Function <Void> Main ()
    Output("Some" & & "Text")
EndFunction
```

Status: Idle | 4 Line(s) | Line: 4, Col: 12 | Zoom: 180%

Lecture Notes

From the Masters course at Uni. Washington

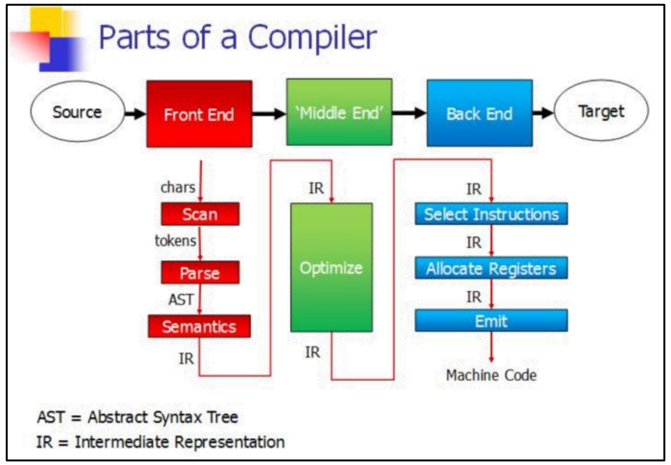




https://courses.cs.washington.edu/courses/csep501/14sp/video/archive/html5/video.html?id=csep501_14sp_1

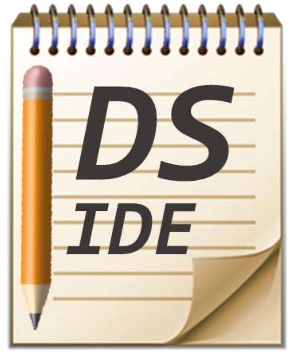
General

- Front end: specific to language being compiled
- Back end: could be used for any language
- <FrontEnd…>
- **Scanner: Source → Tokens**
- Parser: Tokens → AST (an AST is a type of IR)
- Semantic Analysis: AST → Check logical sense (E.g. [int x = 42 + true] would be invalid)
- <BackEnd…>
- Optimisation: Several phases of code improvements (completely reorganises AST)
- Code Generation: Instruction selection & register allocation – generates MachineCode



Improvements for Me

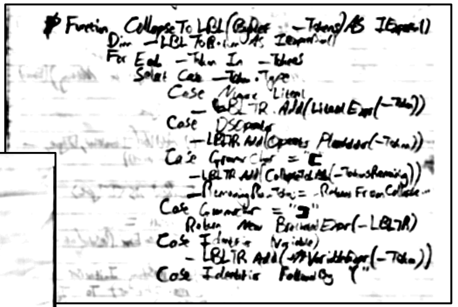
- Token.Value should be Token.Lexeme
- My "*Parser*" should really be called a "Scanner"
- My "*Lexing*" stage is really "Parsing"



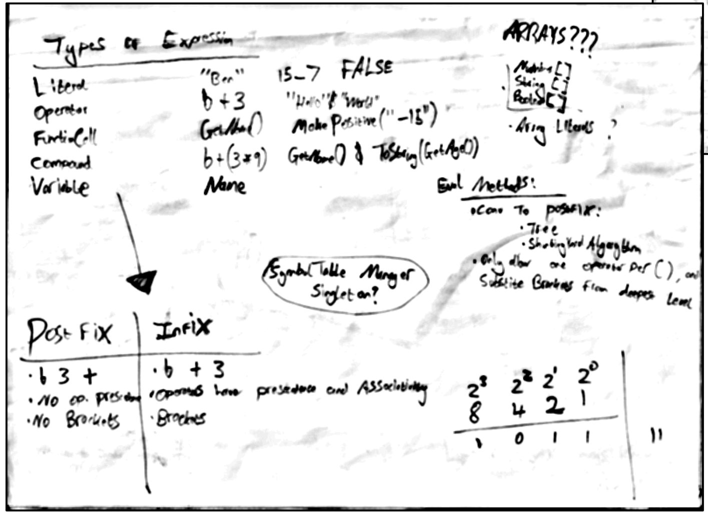
String > Lexeme > Grapheme

Draw Planning

Misc. Planning (typed up elsewhere herein)



```
Function CollapseToLBL(ByRef _Tokens) AS IExpression()
    Dim _LBL_To_Return AS IExpression()
    For Each _Token In _Tokens
        Select Case _Token.Type
            Case Literal
                _LBLTR.Add(LiteralExpr(_Token))
            Case DSOperator
                _LBLTR.Add(OperatorPlaceholder(_Token))
            Case GrammarChar = "["
                _LBLTR.Add(CollapseToLBL(_TokensRemaining))
                _RemainingTokens = _Return From Collapse
            Case GrammarChar = "]"
                Return New BracketedExpr(_LBLTR)
            Case Identifier (Variable)
                _LBLTR.Add(VariableExpr(_Token))
            Case Identifier FollowedBy "("
```



Types of Expression

Literal	"Ben"	15_7 FALSE
Operator	b + 3	"Hello" & "World"
FunctionCall	GetAge()	MakePositive("-15")
Compound	b+(3*9)	GetAge() & ToString(GetAge())
Variable	Name	

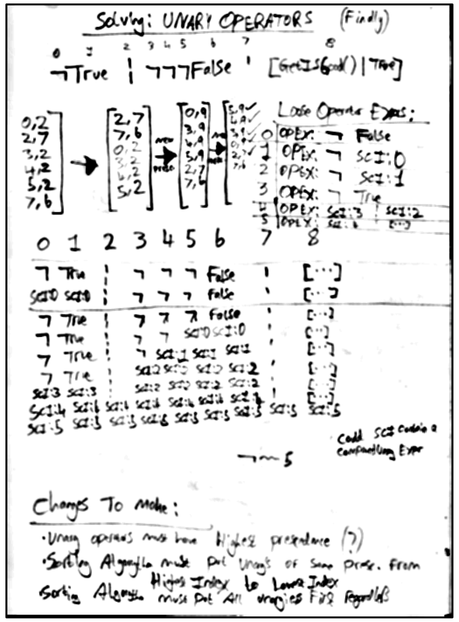
ARRAYS???

Eval Methods:
- Convert To POSTFIX:
  - Tree
  - Shunting Yard Algorithm
- Only allow one operator per ( ), and Substitute Brackets from deepest level

Symbol Table Manager Singleton?

Post Fix	InFix
· b 3 +	· b + 3
· No op. precedence	· Operators have precedence and Associativity
· No Brackets	· Brackets

```
2³ 2² 2¹ 2⁰
8  4  2  1
1  0  1  1    11
```



Solving: UNARY OPERATORS (Finally)

```
0 1 2 3 4 5 6 7         8
¬True  |  ¬¬¬False  |  [GetIsGood() | True]
```

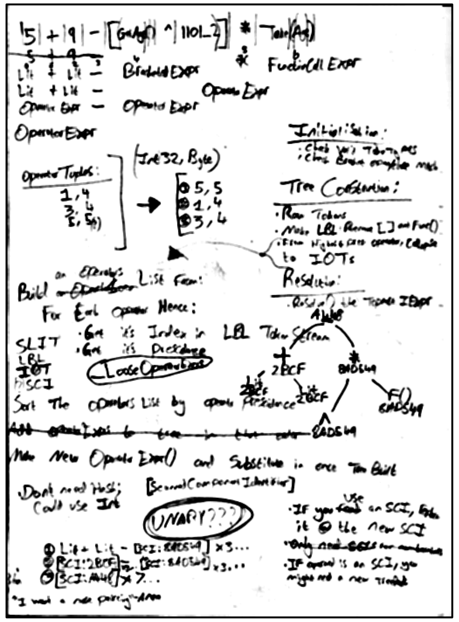
Loose Operator Exprs:

```
0 OPEX: ¬   False
1 OPEX: ¬   SCI:0
2 OPEX: ¬   SCI:1
3 OPEX: ¬   True
4 OPEX: SCI:3  SCI:2
```

```
0   1    2 3 4 5    6    7    8
¬ True   ¬ ¬ ¬ False      [...]
```

¬¬¬5

Changes To Make:
- Unary operators must have Highest precedence (?)
- Sorting Algorithm must put Unary of Same Prec. from Highest Index to Lower Index
- Sorting Algorithm must put All Unaries First Regardless



```
5 + 9 - [GetAge() ^ 1101_2] * Tale(Age)
```

Bracketed Expr — FunctionCall Expr — Operator Expr

Operator Tuples: (Int32, Byte)

Initialisation:
- Tree Construction:
- Row Tokens
- Make LBL
- Resolution:

Build an ~~Ordered~~ List from:
For Each Operator Hence:
- Get its Index in LBL
- Get its Precedence

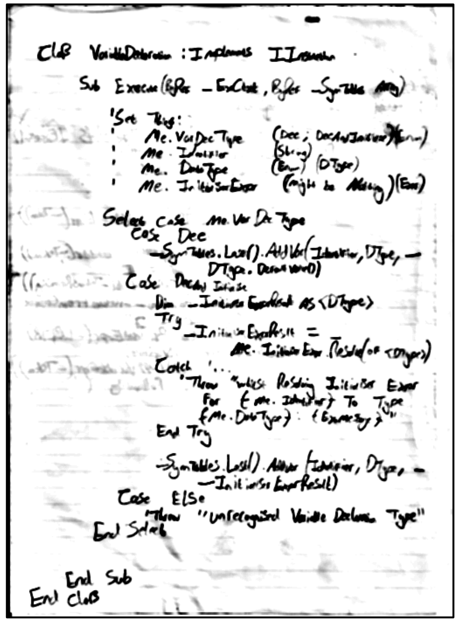
LooseOperatorExpr

Sort the Operators List by operator Precedence

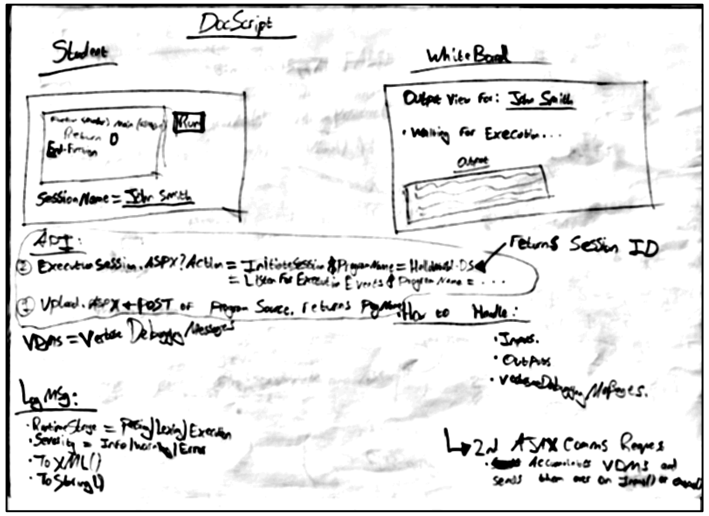
Make New OperatorExpr() and Substitute in once Tree Built

Don't need Hash; Could use Int

UNARY???



```
Class VariableDeclaration : Implements IInstruction
    Sub Execute(ByRef _ExecCtx, ByRef _SymTables As...)
        'Set Things:
        '   Me.VarDecType
        '   Me.Identifier
        '   Me.DataType
        '   Me.InitialiserExpr
        Select Case Me.VarDecType
            Case Decl
                _SymTables.Local().AddVar(_Identifier, DType, DType.DefaultValue())
            Case DeclInitialise
                Dim _InitialiserExprResult AS <DType>
                Try
                    _InitialiserExprResult = Me.InitialiserExpr.Resolve(Of <DType>)
                Catch
                    Throw "while Resolving Initialiser Expr"
                End Try
                _SymTables.Local().AddVar(_Identifier, DType, _InitialiserExprResult)
            Case Else
                Throw "Unrecognised Variable Declaration Type"
        End Select
    End Sub
End Class
```



DocScript

Student — WhiteBoard

Output View For: John Smith
- Waiting For Execution...

SessionName = John Smith

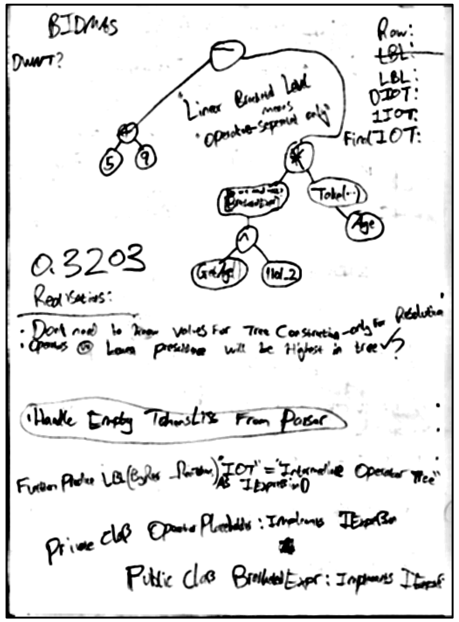
API:
1. ExecutionSession.ASPX?Action = InitiateSession & ProgramName = HelloWorld.DS — Returns Session ID
2. Upload.ASPX + POST of Program Source, returns ...

How to Handle:
- Inputs
- Outputs
- VerboseDebugging Messages

VDMs = Verbose Debugging Messages

LogMsg:
- RuntimeStage = Parsing | Lexing | Execution
- Severity = Info | Warning | Error
- ToXML()
- ToString()

L→ 2nd AJAX Comms Request: Accumulates VDMs and Sends them over on Input() or ...



BIDMAS

Row: ~~LBL~~ LBL, OIOT, IIOT, FinalIOT

"Linear Bracketed List" means "Operator-Separated only"

0.3203

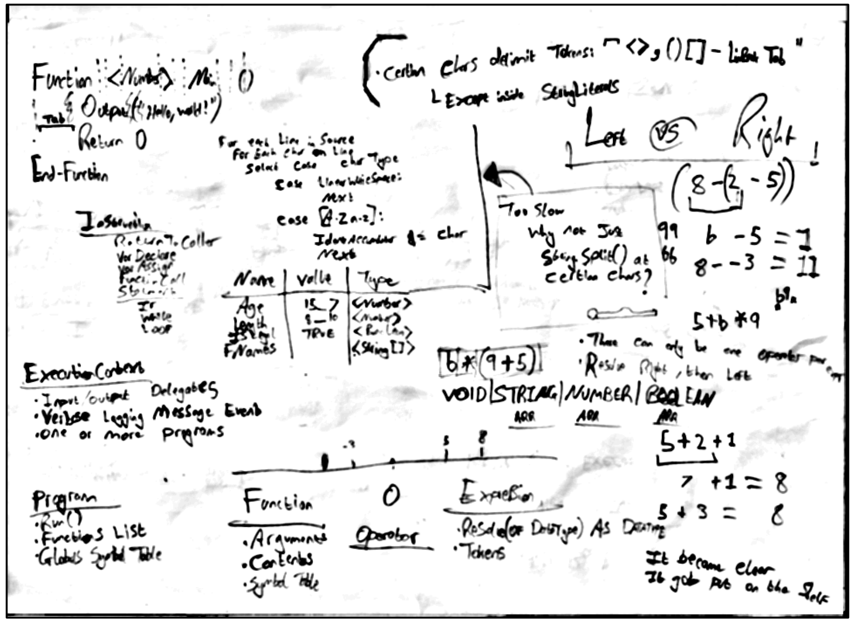
Realisations:
- Don't need to know values for Tree Constructing — only for Resolution
- Operators @ Lower positions will be Highest in tree ✓

Handle Empty TokensLISTS From Parser

Function Produce LBL(ByRef _Tokens) "IOT" = "Intermediate Operator Tree" AS IExpression()

Private Class OperatorPlaceholder : Implements IExpression

Public Class BracketedExpr : Implements IExpression



```
Function <Number> Main ()
    Output("Hello, World!")
    Return 0
End-Function
```

Certain Chars delimit Tokens: " <>, ()[] - LineBreak Tab "
L Escape inside StringLiterals

Left VS Right!

```
(8 - (2 - 5))
b -5 = 1
8 - -3 = 11
```

Too Slow
Why not Just String.Split() at certain chars?

For each Line in Source
  For Each Char on Line
    Select Case char Type
      Case LineWhiteSpace: Next
      Case [A-Z a-z]: IdentAccumulator &= char Next

Instructions:
- Return To Caller
- Var Declare
- Var Assign
- Func. Call
- Statements: If, While, Loop

Name	Value	Type
Age	15_7	<Number>
Length	8_10	<Number>
IsStupid	TRUE	<Boolean>
FNames		<String[]>

Execution Contexts
- Input/Output Delegates
- Verbose Logging Message Events
- One or more Programs

b * (9 + 5)

VOID | STRING | NUMBER | BOOLEAN

5 + 2 + 1
7 + 1 = 8
5 + 3 = 8

- There can only be one operator per ...
- Resolve Right, then Left

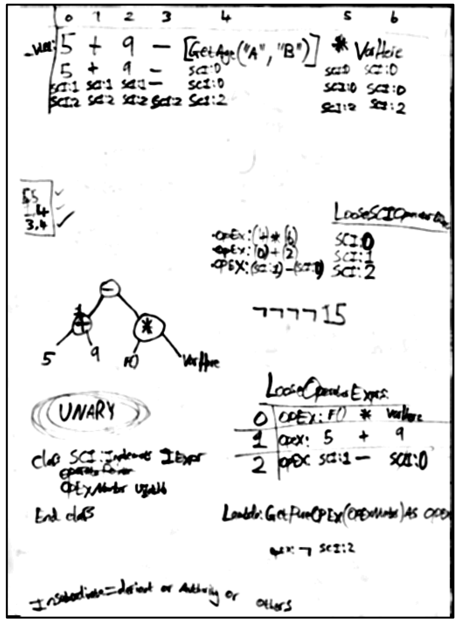
It became clear It goes RHS on the Left

Program
- Run()
- Functions List
- Globals Symbol Table

Function
- Arguments
- Contents
- Symbol Table

Operator

Expression
- Resolve(Of DataType) As DataType
- Tokens



```
0 1 2 3        4          5  6
5 + 9 - [GetAge("A", "B")] * Value
```

UNARY

Class SCI : Implements IExpr
End Class

LooseSCIOperatorExpr

LooseOperatorExprs

0	OPEX: FC	*	Value
1	OPEX: 5	+	9
2	OPEX: SCI:1	-	SCI:0

¬¬¬¬15

Considerations

What I must henceforth consider

ToDo...

- **Make DSFunction take in IDataValues for Arguments, not IExpressions**
- Add ToString() Methods to ReturnStatus, ExecutionResult, DSString, etc…
- Check that Void@ raises a syntax error
- Implement BasedNumbers for DSNumber
- Make the Catch one-liners concat in the TypeName of the TypeOf _Ex
- Make DSWebException concat. in the Full URL
- Finish the DSOperators and make each of them have a Try which [Throw]s [DSOperatorExecutionException]s

Possible Features...

- [WinIDE] Insert Snippets
- [WinIDE] Browse available BuiltInFunctions
- $ LoopIteration Variable or BuiltInFunction Herefor
- Namespaces
- Next in Whiles and Loops
- Comments which start mid-way through a line #Like This...
- Block Comments (###)
- Array Literals {1,2,3,4,5}
- Constants (different EntryType in SymbolTable)

To add to CS-Writeup

- [Prototype Review] Sections with {Screenshots, "What has been done", "How it has been tested", "How it relates to the problem-breakdown", "How it meets the User Requirements" (WHICH CREITERIA ARE THEREBY MET?), "Problems, and changes made as a result of the prototype"}
- Explain how my code is: Modular, Well-Structured, Uses good naming conventions
- Have clearly-delimited **"TESTING"** Sections.

DocScript Exit Codes

0	No Errors Occurred
Non-Zero	An Error Occurred

Compilation [0]

Compiling DocScript programs to .NET exes

↓ Not needed; ExeCxt accessible at .NET Runtime.

Compilation Execution-Context? Compilation-Context?

Program.Compile("HelloWorld.EXE") instead of Program.Run(_CLAs)

Have a "Compilation Options" dialog with options for: {the EXE-Icon, the EXE Name, the CompilationContext for CLI or GUI, Compile-to-DLL}

```
Function <Void> Main ()
    <Number> _Test : Input("Enter Test Number")
    Output(_Test & " is a Prime: " & IsAPrime(_Test))
EndFunction

Function <Boolean> IsAPrime (<Number> _Test)

    <Number> _I : 2
    While ( Maths_LessThan(_I * _I, _Test) | [[_I * _I] = _Test] )
        If ([_Test % _I] = 0)
            Return False
        EndIf
        _I : _I + 1
    EndWhile

    Return True

EndFunction
```

```
Dim _Test As Double = InputBox("Enter Test Number")
```

Option Explicit On
Option Infer On
Option Strict On

```
DocScript.Compilation.RunBIF("Maths_LessThan", _I * _I, _Test)
#Where:
- RunBIF() Takes in: String, ParamArray Object
- RunBIF() Returns: Object (Double/(), String/(), Boolean/(), Nothing)
```

```
<Assembly: System.Reflection.AssemblyTitle("My EXE")>
<Assembly: System.Reflection.AssemblyDescription("Description")>
<Assembly: System.Reflection.AssemblyCompany("Author")>
<Assembly: System.Reflection.AssemblyProduct("Product")>
<Assembly: System.Reflection.AssemblyCopyright("BRAND (C) 2022")>
<Assembly: System.Reflection.AssemblyTrademark("BRAND (TM)")>
<Assembly: System.Runtime.InteropServices.ComVisible(False)>
<Assembly: System.Reflection.AssemblyVersion("1.3.1.0")>
<Assembly: System.Reflection.AssemblyFileVersion("2.1.0.4")>
<Assembly: System.Resources.NeutralResourcesLanguageAttribute("en-GB")>
```

```
Function <Void> Main ()
    Output("Hello, World!")
    Return
EndFunction
```

⬇

Compile with:
```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\vbc.exe "HelloWorld.VB" /out:"HelloWorld.exe" /win32icon:"EXEIcon.ico" /target:exe
```

Or winexe (to not AllocConsole for the process)

```
Module Program

        Sub Main()
                MsgBox("Hello, World")
                Return
        End Sub

End Module
```

To Get Compiler Paths:
```
dir %WINDIR%\Microsoft.NET\Framework64\csc.exe /s /b
```

Compilation [1] (Translation)

Translation

DocScript.Translation
<DocScript.Translation.TranslationTarget> DSVisualBasicDotNETTranslator
ITranslatableToVB

DocScript:	**VisualBasic .NET:**
#Comment	(Not translated)
<String> Name	Dim Name As Global.System.String or DocScript.<…>.DSString (?)
<String> Name : "Ben"	Dim Name As Global.System.String = "Ben"
Name : "Ben"	Name = "Ben"
SayHello()	SayHello()
SayHello(Name)	SayHello(Name)
Return	Return
Return 0	Return 0
Function <Void> SayHello ()*	Function SayHello() As DSVoid
Function <Void> SayHello (<String> Name)*	Function <Void> SayHello (<String> Name)*
EndFunction*	End Function
If (True)	If (True)
Else	Else
EndIf	End If
While (True)	While (True)
EndWhile	End While
Loop (10)	Loop (10)
EndLoop	End Loop
* = Cannot appear within a Function Body	* = Cannot appear within a Function Body