



**Keywords and Functions**  
**Reference Guide**

**Document Last Updated: 21<sup>st</sup> July 2010**

# Vaakya Keywords and Functions Reference Guide

## Table of Contents

Introduction .....	6
Audience .....	6
Terminology and Notation .....	6
Variable Declarations .....	6
Vaakya Keywords and Built-In Functions .....	6
Default DataTypes .....	6
Keyword Limitations.....	6
String Functions .....	6
TextLength.....	6
TextUpper .....	6
TextLower .....	6
TextEmpty .....	6
TextTrim .....	6
TextCharacter.....	6
TextGetByteValue.....	6
TextGetCharCount .....	6
TextFindChar .....	6
TextFind .....	6
TextReplaceChar .....	6
TextReplace.....	6
TextParseToken .....	6
TextParseBoundToken.....	6
TextParseDate .....	6
TextParseDateYmd .....	6
TextParseTime .....	6
TextParseNumber .....	6
TextParseKeyValue .....	6
TextFill .....	6
TextPrefix .....	6
TextPostfix .....	6
TextInfix .....	6
TextPart.....	6
TextAlign .....	6
TextMatch .....	6
TextPartMatch.....	6
TextPartFind .....	6
TextContains .....	6
TextAdvance.....	6
Control Keywords .....	6
Loop.....	6
While.....	6
If .....	6
Skip .....	6
Switch Case.....	6
Break .....	6
Return .....	6
Memory Handling Keywords .....	6
New/NewRecset .....	6
Free .....	6
Database Utilities .....	6
VdsDomainCompact.....	6
VdsDomainReIndex .....	6

## Vaakya Keywords and Functions Reference Guide

Database Functions.....	6
VdsSave .....	6
VdsModify .....	6
VdsDelete .....	6
VdsGetRecord .....	6
VdsFindRecord .....	6
VdsGetTable .....	6
VdsGetRecordCount .....	6
VdsGetCluster .....	6
VdsFindCluster .....	6
VdsFetchMore .....	6
VdsFetchClose .....	6
Entity Functions .....	6
VdsAddEntity .....	6
VdsClearEntity .....	6
VdsDeleteEntity .....	6
VdsSaveEntityTran .....	6
VdsModifyEntityTran .....	6
VdsDeleteEntityTran .....	6
VdsGetEntityRecord .....	6
VdsFindEntityRecord .....	6
VdsGetEntityTable .....	6
VdsGetEntityRecordCount .....	6
VdsGetEntityCluster .....	6
VdsFindEntityCluster .....	6
JTV Functions.....	6
JtvFindRecord .....	6
Recset Functions .....	6
RecsetAppend .....	6
RecsetCount .....	6
RecsetFetch.....	6
RecsetFind .....	6
RecsetFilter .....	6
RecsetSplit .....	6
RecsetOrder .....	6
RecsetAggregate .....	6
RecsetGroup.....	6
RecsetDuplicate .....	6
Array Functions .....	6
ArrayAdd .....	6
ArrayGet .....	6
ArrayGetLast .....	6
ArrayCount.....	6
Set Functions .....	6
SetAdd .....	6
SetDelete.....	6
SetFetch .....	6
SetFetchInOrder.....	6
SetContains.....	6
SetEqual .....	6
SetCount.....	6
SetJoin .....	6
SetUnion.....	6
SetIntersect .....	6
Dict Functions .....	6
DictAdd.....	6
DictFind .....	6

## Vaakya Keywords and Functions Reference Guide

DictFindNext .....	6
DictReplace .....	6
DictCount .....	6
Stream Functions .....	6
StreamOpen .....	6
StreamPrint .....	6
StreamOpenFile .....	6
StreamReadOpenFile .....	6
StreamInfo .....	6
StreamWrite .....	6
StreamAppend .....	6
StreamRead .....	6
StreamMapContent .....	6
StreamFind .....	6
StreamScanBoundToken .....	6
StreamCopy .....	6
StreamTruncate .....	6
StreamSave .....	6
StreamClose .....	6
StreamFetchInit .....	6
StreamFetchNext .....	6
StreamFetchOver .....	6
TagParserInit .....	6
TagParserLocateElement .....	6
TagParserGetElementContent .....	6
TagParserGetElementAttribute .....	6
TagParserGetElementAttributeList .....	6
TagParserClose .....	6
Utilities .....	6
Zip .....	6
UnZip .....	6
Encrypt .....	6
Decrypt .....	6
Encypher .....	6
Decypher .....	6
EncodeB64 .....	6
DecodeB64 .....	6
HttpEscape .....	6
HttpUnEscape .....	6
Ascii2Hex .....	6
Hex2Ascii .....	6
GetCurrentTime .....	6
GetCurrentTimeX .....	6
GetCurrentTimeFormatted .....	6
GetCurrentTimeXFormatted .....	6
FileExists .....	6
FileDelete .....	6
DirectoryCreate .....	6
DirectoryInfo .....	6
DirectoryDelete .....	6
StoreGlobalPointer .....	6
FindGlobalPointer .....	6
RemoveGlobalPointer .....	6
IpcRequest .....	6
HttpRequest .....	6
JsonRequestToSet .....	6
JsonRequestToDict .....	6

## Vaakya Keywords and Functions Reference Guide

JsonRequestToObject .....	6
JsonRequestToObjectSet .....	6
SetToJsonResponse .....	6
DictToJsonResponse .....	6
ObjectToJsonResponse .....	6
RecsetToJsonResponse .....	6
<b>Math Functions</b> .....	<b>6</b>
Abs .....	6
Floor .....	6
Ceil .....	6
Sign .....	6
Sqrt .....	6
Exp .....	6
Log .....	6
Log10 .....	6
Sin .....	6
Cos .....	6
Tan .....	6
Asin .....	6
Acos .....	6
Atan .....	6
Round .....	6
Trunc .....	6
Rand .....	6
Min .....	6
Max .....	6
Mod .....	6
Pow .....	6
<b>Misc Functions</b> .....	<b>6</b>
DaysBetween .....	6
FormDate .....	6
DaysInMonth .....	6
Concat .....	6
ValueInWords .....	6
EnumValue .....	6
EnumLabel .....	6
DayofWeek .....	6
DayName .....	6
TextRemoveLineChar .....	6
ConsolePrint .....	6
BitSet .....	6
BitClear .....	6
BitTest .....	6
BitFlip .....	6
SerialPortRead .....	6
SerialPortWrite .....	6
<b>Web Functions</b> .....	<b>6</b>
ComposerInit .....	6
ComposerSubstituteRecord .....	6
ComposerSubstituteVariable .....	6
ComposerSubstituteBlock .....	6
ComposerSave .....	6
ComposerSend .....	6
WebResponseInit .....	6
WebResponsesend .....	6
ConvertMimeDataToRecord .....	6
<b>Communication Functions</b> .....	<b>6</b>

## Vaakya Keywords and Functions Reference Guide

MailBoxInit .....	6
MailDespatch .....	6
MailCheck .....	6
MailReceive .....	6
SMSBoxInit .....	6
SmsDespatch.....	6
SmsReceive.....	6

## Vaakya Keywords and Functions Reference Guide

### Introduction

This document covers all the built in functions and keywords of Vaakya 2.1 version for developing Browser based Web Applications.

### Audience

Developers/Programmers should refer to this document to learn all keywords and functions of Vaakya programming language.

### Terminology and Notation

#### **Notation for syntax specification**

- Contents enclosed in “[ ]” represent optional entries
- “ | ” Symbol is used to indicate “or”
- Capital lettered words represent keywords/functions
- Words in *italics* represent syntactic constructs.
- Temporary variable, Instance Variable and Object Variable are used contextually and mean the same.
- NA – Not Applicable

### Variable Declarations

Variable declarations require

- Datatype
- storage size
- variable type
- data structure required to store the contents of a variable

## Vaakya Keywords and Functions Reference Guide

### Default DataTypes

SNo	Data type	Default Size (Bytes)	Remarks
1	Text	1	Primitive DataType
2	TextDate	8	Primitive DataType
3	TextTime	6	Primitive DataType
4	Tiny	1	Primitive DataType
5	Short	2	Primitive DataType
6	Int	4	Primitive DataType
7	Long	4	Primitive DataType
8	Byte	1	Primitive DataType
9	Byte2	2	Primitive DataType
10	Byte4	4	Primitive DataType
11	ByteX	8	
12	Bool	4	Primitive DataType
13	Enum	4	Enumerator
14	Date	4	Primitive DataType
15	Time	4	Primitive DataType
16	TimeX	8	Primitive DataType
17	Float	8	Primitive DataType
18	Double	8	Primitive DataType
19	Image		Unstructured DataType
20	Video		Unstructured DataType
21	Audio		Unstructured DataType
22	Binary		Unstructured DataType
23	Stream		Structured DataType
24	Recset		Structured DataType.Not allowed in Objects
25	Array		Structured DataType.Not allowed in Objects
25	Dict		Structured DataType.Not allowed in Objects
26	Set		Structured DataType .Not allowed in Objects

## Vaakya Keywords and Functions Reference Guide

### Keyword Limitations

1. TextTrim
  - ✓ Text Trim only trims the starting spaces in the Destination pointer, but returns the length of alphanumeric characters removing the trailing spaces.
2. Text Data Type
  - ✓ Text Data Type for Upper case and Lower case is not available. Users have to handle the same using TextUpper or TextLower Functions in Vaakya Proc

### Vaakya Keywords and Built-In Functions

#### String Functions

##### **TextLength**

- ✓ Function Brief  
Function to find the length of a String. Returns Length of a String.
- ✓ Function Syntax  
varReturn = TextLength(SourceString)
- ✓ Function Argument
  1. Source String
- ✓ Example  
Int                    varReturn  
varReturn        = TextLength("Vaakya")  
varReturn will contain 6.

##### **TextUpper**

- ✓ Function Brief  
Function to convert a Text String to Uppercase.
- ✓ Function Syntax  
Output = TextUpper(SourceString)
- ✓ Function Argument
  1. Source String
- ✓ Example  
Text            @OutputString  
OutputString = TextUpper("Vaakya")  
  
OutputString will contain "VAAKYA".

## Vaakya Keywords and Functions Reference Guide

### **TextLower**

✓ [Function Brief](#)

Function to convert a Text String to Lowercase.

✓ [Function Syntax](#)

Output = TextLower(SourceString)

✓ [Function Argument](#)

1. Source String

✓ [Example](#)

```
Text           @OutputString
OutputString = TextLower("Vaakya")

OutputString will contain "vaakya"
```

### **TextEmpty**

✓ [Function Brief](#)

Function to check if a Text String is Empty. Returns 1 if String is Empty else Returns 0.

✓ [Function Syntax](#)

ReturnValue = TextEmpty(SourceString)

✓ [Function Argument](#)

1. Source String

✓ [Example](#)

```
Bool           ReturnValue
ReturnValue = TextEmpty("VAAKYA")
```

### **TextTrim**

✓ [Function Brief](#)

Function to Trim white spaces before and after a Text String. Returns the length of the trimmed text.

✓ [Function Syntax](#)

varOutput = TextTrim(SourceString)

✓ [Function Argument](#)

1. Source String

✓ [Example](#)

```
Int           varOutput
Text          SourceString[15] SourceString = "    Vaakya    "

varOutput = TextTrim(SourceString)

varOutput will return 6.
```

## Vaakya Keywords and Functions Reference Guide

### **TextCharacter**

✓ [Function Brief](#)

Function to extract the ASCII value of a character from a given Text String, based on an Index.

✓ [Function Syntax](#)

Output = TextCharacter(src[idx])

✓ [Function Arguments](#)

1. Source String
2. Position Index

✓ [Example](#)

```
Int         varOutput
Text        @varText
varText = Concat("Vaakya")
varOutput = TextCharacter(varText[1])
```

varOutput will return 97 which is the ASCII value for "a".

### **TextGetByteValue**

✓ [Function Brief](#)

Function to extract the ASCII value of a character from a given Text String, based on an Index.

✓ [Function Syntax](#)

OutputChar = TextGetByteValue(source,idx)

✓ [Function Arguments](#)

1. Source String
2. Position Index

✓ [Example](#)

```
Int         varByte
varByte = TextGetByteValue("Vaakya",1)
```

varByte will return 97.

### **TextGetCharCount**

✓ [Function Brief](#)

Function to find the number of occurrences of a character in a Text String

✓ [Function Syntax](#)

Output = TextGetCharCount(source,char)

✓ [Function Arguments](#)

1. Source String
2. Find Character

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
Int      varCount
varCount = TextGetCharCount("Vaakya","a")

varCount will return 3.
```

## **TextFindChar**

### ✓ [Function Brief](#)

Checks the occurrence of a single character in a Text String and returns the text from the first occurrence of the character.

### ✓ [Function Syntax](#)

```
Output = TextFindChar(src, char)
```

### ✓ [Function Arguments](#)

1. Source String
2. Find Character

### ✓ [Example](#)

```
Text      @varOutput
varOutput = TextFindChar("Vaakya","k")
varOutput will contain "kya"
```

## **TextFind**

### ✓ [Function Brief](#)

Function to find one string within another. Returns the text from the first occurrence of the specified match.

### ✓ [Function Syntax](#)

```
Output = TextFind(SourceString, findstring,MatchFlag)
```

### ✓ [Function Arguments](#)

1. Source String
2. Match String
3. Match Flag
  - a. TEXT\_MATCH\_ANYWAY – For Case InSensitive Match.
  - b. TEXT\_MATCH\_EXACT - For Case Sensitive Match.

### ✓ [Example](#)

```
Text      @varInput, @varOutput, @varMatchText
varInput   = "vaakya tech"
varMatchText = "KYA"
varOutput  = TextFind(varInput, varMatchText,TEXT_MATCH_ANYWAY)

varOutput will return "kya tech".
```

## **TextReplaceChar**

### ✓ [Function Brief](#)

Function to find and replace a character in a Text String with another character.

### ✓ [Function Syntax](#)

```
Output = TextReplaceChar(SourceString, TargetChar, ReplaceChar)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Arguments](#)

1. Source String
2. Character to be Replaced
3. Character to be Replaced With

### ✓ [Example](#)

```
Bool    varOutput
Text    @varInput, @varTargetChar, @varReplaceChar
```

```
varInput      = "Vaakya"
varTargetChar = "a"
varReplaceChar = "m"
```

```
varOutput = TextReplaceChar(varInput, varTargetChar, varReplaceChar)
varOutput will return 1 if the character has been replaced.
VarInput will contain the String: "Vmmkym"
```

## **TextReplace**

### ✓ [Function Brief](#)

This will replace a group of characters in a Text String, with another set of characters.

### ✓ [Function Syntax](#)

```
Output = TextReplace(SourceString, TargetChar(s), ReplaceChar(s))
```

### ✓ [Function Arguments](#)

1. Source String
2. Character(s) to be Replaced
3. Character(s) to be Replaced With

### ✓ [Example](#)

```
Text          @varOutput, @varInput, @varTarget, @varReplaceChar
varInput      = "vaakya"
varTarget     = "vaak"
varReplaceChar = "test"
varOutput     = TextReplace(varInput, varTarget, varReplaceChar)
```

```
varOutput will contain "testya".
```

## **TextParseToken**

### ✓ [Function Brief](#)

This will parse the text and returns individual tokens based on a delimiter in the text.

### ✓ [Function Syntax](#)

```
Output = TextParseToken(SourceString, Delimiter, Advance)
```

### ✓ [Function Arguments](#)

1. Source String
2. Delimiter
3. Advance
  - a. 0 – To terminate the Parser.
  - b. 1 – Advances the Parser to the next Token.

### ✓ [Example](#)

```
Text          @varFirstStr, @varOutput, @varDelimiter, @varMapStr
varFirstStr   = "Vaakya, Parse, Token"
```

## Vaakya Keywords and Functions Reference Guide

```
varMapStr      = varFirstStr
varDelimiter   = ","
varOutput      = TextParseToken(varMapStr, varDelimiter, 0)
```

varOutput will return "Vaakya".

To parse all the tokens, please write the above TextParseToken inside a loop.

### **Note:**

- In the above example TextParseToken cannot be used on "varFirstStr". If the same pointer variable is used, while parsing the pointer position would have advanced hence replacing the base pointer information
- TextParse function advance pointer is available for pointers only which are Ethereal.

## **TextParseBoundToken**

### ✓ [Function Brief](#)

This will parse the text and returns individual tokens based on a delimiter pair in the text.

### ✓ [Function Syntax](#)

```
Output = TextParseBoundToken(SourceString, DelimiterPair, Advance)
```

### ✓ [Function Arguments](#)

1. Source String
2. Delimiter Pair
3. Advance
  - a. 0 – To terminate the Parser.
  - b. 1 – Advances the Parser to the next Token.

### ✓ [Example](#)

```
Text          @varFirstStr, @varSecondStr, @varOutput

varFirstStr   = "{Vaakya}{Parse}{Token}"
varSecondStr  = varFirstStr
varOutput     = TextParseBoundToken(varSecondStr, "{ }", 1)

varOutput will return "Vaakya".
```

### **Note:**

- In the above example TextParseBoundToken cannot be used on "varFirstStr". If the same pointer variable is used, while parsing the pointer position would have advanced hence replacing the base pointer information
- TextParse function advance pointer is available for pointers only which are Ethereal.

## **Note on Date Functions**

- If you print the Output Date using 'ConsolePrint', the format will always be "YYYYMMDD"

## **TextParseDate**

### ✓ [Function Brief](#)

Function to Parse a Delimited Text and extract Date.

### ✓ [Function Syntax](#)

```
Output = TextParseDate(SourceString, Delimiter, Advance)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Arguments](#)

1. Source String
2. Delimiter
3. Advance
  - a. 0 – To terminate the Parser.
  - b. 1 – Advances the Parser to the next Token

### ✓ [Example](#)

```
Text          @varInput, @varTemp
TextDate      varOutputDate

varInput      = "01-12-2008,15:12:10"
varTemp       = varInput
varOutputDate = TextParseDate(varTemp, ",", 1)

varOutputDate will return "01122008".
```

#### **Note:**

- In the above example TextParseDate cannot be done for "varInput". If the same pointer variable were used, while parsing the pointer position would have advanced hence replacing the base pointer information.
- TextParse function advance pointer is available only for pointers which are Ethereumal.

## **TextParseDateYmd**

### ✓ [Function Brief](#)

This will return a date from a text in yyymmdd format based on a delimiter in the text.

### ✓ [Function Syntax](#)

```
Output = TextParseDateYmd(SourceString, Delimiter, Advance)
```

### ✓ [Function Arguments](#)

1. Source String
2. Delimiter
3. Advance
  - a. 0 – To terminate the Parser.
  - b. 1 – Advances the Parser to the next Token

### ✓ [Example](#)

```
Text          @varInput, @varTemp
TextDate      varOutputDate
varInput      = "01-12-2008,15:12:10"
varTemp       = varInput
varOutputDate = TextParseDateYmd(varTemp, ",", 1)

varOutputDate will return "20081201".
```

#### **Note:**

- In the above example TextParseDateYmd cannot be done for "varInput". If the same pointer variable were used, while parsing the pointer position would have advanced hence replacing the base pointer information.
- TextParse function advance pointer is available only for pointers which are Ethereumal.

## Vaakya Keywords and Functions Reference Guide

### **TextParseTime**

#### ✓ [Function Brief](#)

Function to Parse a Delimited Text and extract Time.

#### ✓ [Function Syntax](#)

```
Output = TextParseTime(SourceString, Delimiter, Advance)
```

#### ✓ [Function Arguments](#)

1. Source String
2. Delimiter
3. Advance
  - a. 0 – To terminate the Parser.
  - b. 1 – Advances the Parser to the next Token

#### ✓ [Example](#)

```
Text           @varInput, @varTemp
TextDate       varOutputDate
TextTime       varOutputTime
```

```
varInput       = "01-12-2008,15:12:10"
varTemp        = varInput
varOutputDate  = TextParseDate(varTemp, ",", 1)
varOutputTime  = TextParseTime(varTemp, ",", 1)
varOutputTime will return "15:12:10".
```

#### **Note:**

- In the above example TextParseTime cannot be done for "varInput". If the same pointer variable were used, while parsing the pointer position would have advanced hence replacing the base pointer information.
- TextParse function advance pointer is available only for pointers which are Ethereal.

### **TextParseNumber**

#### ✓ [Function Brief](#)

Function to Parse a Delimited Text and Extract a Number.

#### ✓ [Function Syntax](#)

```
Output = TextParseNumber(SourceString, Delimiter, Advance)
```

#### ✓ [Function Arguments](#)

1. Source String
2. Delimiter
3. Advance
  - a. 0 – To terminate the Parser.
  - b. 1 – Advances the Parser to the next Token

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
Text      @varInput, @varTemp, varDelimiter[1]
Double    varOutput

varInput   = Concat("1.50, 5.124, 9.854")
varTemp    = varInput
varDelimiter = ","
varOutput  = TextParseNumber(varTemp, varDelimiter, 1)
varOutput will return "1.50".
```

#### **Note:**

- In the above example TextParseNumber cannot be done for "varInput". If the same pointer variable were used, while parsing the pointer position would have advanced hence replacing the base pointer information
- TextParse function advance pointer is available for pointers only which are Ethereal.

## **TextParseKeyValue**

### ✓ [Function Brief](#)

Function to parse and extract value from a delimited Key Value pair based on a Key.

### ✓ [Function Syntax](#)

Output = TextParseKeyValue(SourceString, Delimiter, Key, Advance)

### ✓ [Function Arguments](#)

1. Source String
2. Delimiter
3. Key Value
4. Advance
  - a. 0 – To terminate the Parser.
  - b. 1 – Advances the Parser to the next Token

### ✓ [Example](#)

```
Text      @varInput, varQuote[1], @varOutput, @varKey, varDelimiter[1]
Text      @varInput1

varQuote   = ` `
varDelimiter = ","
varInput   = Concat("name=", varQuote, "TestName", varQuote)
varInput1  = Concat(varInput, varDelimiter, "email=", varQuote, "test1@mail.com",
varQuote)

varKey     = Concat("name")
varOutput  = TextParseKeyValue(varInput1, varDelimeter, varKey, 1)

varOutput will return "TestName".
```

#### **Note:**

- In the above example TextParseKeysValue cannot be done for "varInput". If the same pointer variable were used, while parsing the pointer position would have advanced hence replacing the base pointer information
- TextParse function advance pointer is available only for pointers which are Ethereal.

## Vaakya Keywords and Functions Reference Guide

### **TextFill**

✓ [Function Brief](#)

Function to fill a Text String with a character from a specific length.

✓ [Function Syntax](#)

Output = TextFill(SourceString, FillChar, Length)

✓ [Function Arguments](#)

1. Source String
2. Character to be Filled
3. Length to be Filled

✓ [Example](#)

```
Text  @varInput, varFillChar[1]
Bool  varOutput
```

```
varInput      = Concat("vaakya")
varFillChar   = "*"
varOutput     = TextFill(varInput, varFillChar, 10)
```

varOutput will return 1. varInput will contain "\*\*\*\*\*"

### **TextPrefix**

✓ [Function Brief](#)

Function to prefix an ASCII character to a Text String.

✓ [Function Syntax](#)

Output = TextPrefix(SourceString, PrefixChar, Length)

✓ [Function Arguments](#)

1. Source String
2. Character to be Prefixed
3. Length

✓ [Example](#)

```
Text      @varInput,@varOutput, varFillChar[1]
Int       varLen
```

```
varInput     = Concat("vaakya")
varLen       = 10
varFillChar  = "*"
varOutput    = TextPrefix(varInput, varFillChar, varLen)
```

varOutput will contain "\*\*\*\*vaakya".

## Vaakya Keywords and Functions Reference Guide

### **TextPostfix**

✓ [Function Brief](#)

Function to add an ASCII character at the end of a Text String.

✓ [Function Syntax](#)

Output = TextPostfix(SourceString, PostfixChar, Length)

✓ [Function Arguments](#)

1. Source String
2. Character to be Suffixed
3. Length

✓ [Example](#)

```
Text      @varInput,@varOutput, varFillChar[1]
Int       varLen

varInput  = Concat("vaakya")
varLen    = 10
varFillChar = "*"
varOutput = TextPostfix(varInput, varFillChar, varLen)

varOutput will contain "vaakya*****".
```

### **TextInfix**

✓ [Function Brief](#)

Function to insert any ASCII character to the source text at a given position based on a specific length.

✓ [Function Syntax](#)

Output = TextInfix(SourceString, InfixChar, Length, TextPosition)

✓ [Function Arguments](#)

1. Source String
2. Character to be Inserted
3. Length

✓ [Example](#)

```
Text  @varInput,@varOutput,varFillChar[1]
Int   varLen, varStartPos

varInput    = Concat("vaakya")
varLen      = 10
varStartPos = 2
varFillChar = "*"

varOutput    = TextInfix(varInput, varFillChar, varLen, varStartPos)

varOutput will return "va***akya".
```

## Vaakya Keywords and Functions Reference Guide

### **TextPart**

✓ [Function Brief](#)

Function returns a portion of the Text String from the Start Position in the string to the specified length.

✓ [Function Syntax](#)

Output = TextPart(SourceString, StartPosition, StringLength)

✓ [Function Arguments](#)

1. Source String
2. Starting Position
3. Length

✓ [Example](#)

```
Text @varInput,@varOutput
Int  varStartPos, varLen

varInput      = "vaakya"
varStartPos   = 2
varLen        = 6
varOutput     = TextPart(varInput, varStartPos, varLen)
varOutput will contain "akya".
```

### **TextAlign**

✓ [Function Brief](#)

This will align a Text to the specified position of the total format size specified.

✓ [Function Syntax](#)

Output = TextAlign(SourceString, FormatWidth, AlignFlag)

✓ [Function Arguments](#)

1. Source String
2. Width
3. Align Flag
  - a. TEXT\_ALIGN\_LEFT – Align Left.
  - b. TEXT\_ALIGN\_RIGHT – Align Right.
  - c. TEXT\_ALIGN\_CENTRE – Centre Align.

✓ [Example](#)

```
Text @varInput,@varOutput
Input FormatWidth

varInput      = "vaakya"
FormatWidth   = 10
varOutput     = TextAlign(varInput,FormatWidth, TEXT_ALIGN_LEFT)

varOutput will return "vaakya  ".
```

## Vaakya Keywords and Functions Reference Guide

### TextMatch

✓ [Function Brief](#)

Function to match two strings of same length.

✓ [Function Syntax](#)

Output = TextMatch(SourceString,matchstring,MatchFlag)

✓ [Function Arguments](#)

1. Source String
2. Match String
3. Match Flag
  - a. TEXT\_MATCH\_ANYWAY - For Case InSensitive Match.
  - b. TEXT\_MATCH\_EXACT - For Case Sensitive Match.

✓ [Example](#)

```
Text @varInput,@varOutput, @varMatch
```

```
varInput      = Concat("vaakya")
varMatch      = Concat("VAAKYA")
varOutput     = TextMatch(varInput, varMatch, TEXT_MATCH_ANYWAY)
```

varOutput will return 1 if the text matches.

### TextPartMatch

✓ [Function Brief](#)

Function to match two strings partially for a given length.

✓ [Function Syntax](#)

Output = TextPartMatch(SourceString,Partstring, strlen,MatchFlag)

✓ [Function Arguments](#)

1. Source String
2. Match String
3. Length to be Matched
4. Match Flag
  - a. TEXT\_MATCH\_ANYWAY - For Case InSensitive Match.
  - b. TEXT\_MATCH\_EXACT - For Case Sensitive Match.

✓ [Example](#)

```
Text          @varInput,@varOutput,@varMatch
Int           varLen
```

```
varInput      = Concat("vaakya")
varLen        = 3
varMatch      = Concat("vaa")
varOutput     = TextPartMatch(varInput, varMatch, varLen, TEXT_MATCH_ANYWAY)
```

varOutput will return 1 if match is true.

## Vaakya Keywords and Functions Reference Guide

### **TextPartFind**

✓ [Function Brief](#)

Function to find a given text in the source text for a specific Length.

✓ [Function Syntax](#)

```
Output = TextPartFind(SourceString, findstring, varLen, MatchFlag)
```

✓ [Function Arguments](#)

1. Source String
2. Match String
3. Length to be Matched
4. Match Flag
  - a. TEXT\_MATCH\_ANYWAY - For Case InSensitive Match.
  - b. TEXT\_MATCH\_EXACT - For Case Sensitive Match

✓ [Example](#)

```
Text      @varInput, @varOutput, @varMatch
Int       varLen
```

```
varInput  = Concat("vaakya technology")
varMatch  = Concat("VAA")
varLen    = 3
varOutput = TextPartFind(varInput, varMatch, varLen, TEXT_MATCH_ANYWAY)
```

varOutput will return "vaakya technology".

### **TextContains**

✓ [Function Brief](#)

Function to find a given text pattern in the source text. If the pattern is found the output will be 1 otherwise 0.

✓ [Function Syntax](#)

```
Output = TextContains(SourceString,pattern)
```

✓ [Function Arguments](#)

1. Source String
2. Match Pattern

✓ [Example](#)

```
Text @varInput,@varPattern
Int  varRet
```

```
varRet    = 0
varInput  = Concat("vaakya")
varPattern = Concat("kya")
```

```
varRet    = TextContains(varInput, varPattern)
```

varRet will return 1.

## Vaakya Keywords and Functions Reference Guide

### TextAdvance

✓ [Function Brief](#)

Text advance will advance the position of a given source text for a specific length and return a text from the advanced position till the end.

✓ [Function Syntax](#)

Output = TextAdvance(SourceString,Advance)

✓ [Function Arguments](#)

1. Source String
2. Position Advance

✓ [Example](#)

```
Text  @varInput,@varOutput
Int   varLen

varLen    = 2
varInput  = Concat("vaakya")
varOutput = TextAdvance(varInput,varLen)

varOutput will contain "akya".
```

### Control Keywords

#### Loop

✓ [Function Brief](#)

'Loop' is similar to 'for or while' function and is applicable only on a SET (not for object or user-defined Type)

✓ [Function Syntax](#)

```
Loop(tmpSet, objectVariable)
{
}
```

#### While

✓ [Function Brief](#)

While' is a simple repetitive function used to repeat for a specific number of times.

✓ [Function Syntax](#)

```
While(Condition)
{
}
```

#### If

✓ [Function Brief](#)

If is conditional function

✓ [Function Syntax1](#)

```
If(Condition)
```

## Vaakya Keywords and Functions Reference Guide

```
{  
}
```

### ✓ [Function Syntax2](#)

```
If(Condition)  
{  
}  
Else If(Condition)  
{  
}  
Else  
{  
}
```

## Skip

### ✓ [Function Brief](#)

Skips the rest of the statements in the block and restarts the block with next iteration or record in "Loop or While" functions.

### ✓ [Function Syntax](#)

```
Skip
```

### ✓ [Example](#)

```
Loop( )  
{  
    If( )  
    {  
        Skip  
    }  
}
```

## Switch Case

### ✓ [Function Brief](#)

This will handle multiple cases based on a value.

### ✓ [Function Syntax](#)

```
Switch(Value)  
{  
    Case 100:  
        Statements  
        Break  
  
    Case "Default":  
        Statements  
        Break  
}
```

If the value is a text, it should be specified within quotes.eg.case 'test'

## Break

### ✓ [Function Brief](#)

Function to Exit a "loop".

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Syntax](#)

Break

## **Return**

### ✓ [Function Brief](#)

Function used to exit a proc or a loop without executing further statements.

### ✓ [Function Syntax](#)

Return

## Memory Handling Keywords

### **New/NewRecset**

#### ✓ [Function Brief](#)

'New' allocates memory for a specific primitive data type, object or user defined type. Variables declared, as pointer should be allocated with memory using New.

#### **Note:**

Some of the Proc functions allocate memory and returns the pointer. In such cases, memory need not be allocated prior to usage. Please refer to the specific proc functions.

#### ✓ [Function Syntax](#)

```
X          = New(datatype,1)
ptrSet = NewRecset(Recset, 1)
```

## **Free**

#### ✓ [Function Brief](#)

Free function can be used to free the memory allocated using New function.

#### ✓ [Function Syntax](#)

Free(Recset/Array/Variables)

## Database Utilities

### **VdsDomainCompact**

#### ✓ [Function Brief](#)

Function to remove deleted records from a Vaakya Database inside the active domain. On successful compaction the function returns 1 else it will return 0.

#### ✓ [Function Syntax](#)

Output =VdsDomainCompact()

### **VdsDomainReIndex**

#### ✓ [Function Brief](#)

Function to Re-create database indexes for the active domain. Returns 1 if Re-index is successful else it will return 0.

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Syntax](#)

Output =VdsDomainReIndex()

## Database Functions

### **VdsSave**

#### ✓ [Function Brief](#)

Function to save a record / recset / cluster into an object.

#### ✓ [Function Syntax](#)

Output = VdsSave(ptrset, trantype)

#### ✓ [Function Arguments](#)

1. RecordSet
2. Transaction Type. One of the following
  - a. VDS\_RECORD\_TRAN  
To save a single Record.
  - b. VDS\_RECSET\_TRAN  
To save a set of Records.
  - c. VDS\_CLUSTER\_TRAN  
To Save a record into a Master Slave Object.

#### ✓ [Example1 \(Saving a Single Record into a Database\)](#)

```
ItemGroup  varGroup           ← Object Variable declaration
Array      @varArray          ← Variable declarations
Int        varRet

varArray   = New(Array,1)      ← Memory Allocation
varGroup   = New(ItemGroup, 1)

varGroup.GroupName = 'Stationaries' ← Assignment
varGroup.MinValue  = 10
ArrayAdd(varArray, varGroup)     ← Adding the record to the array
VarRet = VdsSave(varArray, VDS_RECORD_TRAN) ← Saves the record in to the object
```

#### [Brief Description](#)

The above example adds a new record "Stationaries / 10" into the ItemGroup Database.

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example2 \(Saving a Set of Records into a Database\)](#)

```
ItemGroup  varGroup, Recset @setGroup    ← Recset declaration
Int        varRet,i                    ← Variable declarations
Text      @varText
Array     @varArray

varArray   = New(Array,1)              ← Memory Allocation
varGroup   = New(ItemGroup, 1)
setGroup   = NewRecset(ItemGroup, 1)

i = 0
While(i<= 2)
{
  varText           = Concat("Group",i)
  varGroup.GroupName = varText          ← Assignment
  varGroup.MinValue = 10
  RecsetAppend(setGroup, varGroup)     ← Append the value to the object set

  i+=1
}

ArrayAdd(varArray, setGroup)           ← Adding the record to the array
varRet = VdsSave(varArray, VDS_RECSET_TRAN) ← Saves the record in to the object
```

#### Brief Description

The above example will add two records into the ItemGroup database with the following details.

1. Group Name : Group 0 / Min Value - 10
2. Group Name : Group 1 / Min Value - 10

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example 3 \(Saving a Record into a Master Slave – Cluster Object\)](#)

```
Int          varRet,   total, i          ← Variable Declarations
Text        @varText, @date

SaleBill    varSale
SaleBillDet varSalesDet                ← Object variable Declarations

SaleBillDet Recset @setSales           ← Recset declaration

Array       @varArray

varArray    = New(Array,1)             ← Memory Allocation
varSale     = New(SaleBill,1)
varSalesDet = New(SaleBillDet,1)
setSales    = NewRecset(SaleBillDet,1)

varSale.BillNo = "BILL0001"           ← Assignments
date           = Concat("16", "-", "07", "-", "2010")
varSale.Date   = TextParseDate(date, ' ', 1)

ArrayAdd(varArray, varSale)
i = 1

While(i <= 2)
{
    varText           = Concat("ItemName",i)
    varSalesDet.ItemName = varText
    varSalesDet.Qty   = 10
    varSalesDet.Rate  = 10
    varSalesDet.Amount = (varSalesDet.Qty * varSalesDet.Rate)
    total             += varSalesDet.Amount
    RecsetAppend(setSales, varSalesDet)
                                ← Saving into RecordSet

    i += 1
}
varSale.Total = total
ArrayAdd(varArray, setSales)
varRet = VdsSave(varArray,VDS_CLUSTER_TRAN)
                                ← Saves the Cluster in to the object
```

#### Brief Description

The above example will add one record into the SaleBill & SaleBillDet database with the following details.

Bill No : BILL0001                      Date : 16-07-2010                      ← SaleBill Object

Item Name	Qty	Rate	Amount
ItemName1	10	10	100
ItemName2	10	10	100

← SaleBillDet Object

← SaleBillDet Object

## Vaakya Keywords and Functions Reference Guide

### **VdsModify**

#### ✓ [Function Brief](#)

Function to modify an existing record / recset / cluster in an object.

#### ✓ [Function Syntax](#)

Output = VdsModify(ptrset, trantype)

#### ✓ [Function Arguments](#)

1. RecordSet
2. Transaction Type. One of the following
  - a. VDS\_RECORD\_TRAN  
To Modify a single Record.
  - b. VDS\_RECSET\_TRAN  
To Modify a set of Records.
  - c. VDS\_CLUSTER\_TRAN  
To Modify a record into a Master Slave Object

#### ✓ [Example 1 \(Modify a Database Record\)](#)

```
ItemGroup  varGroup           ← Object Variable Declaration
Int        varRe              ← Variable Declarations
Text      @objName, @branchName, @key
Array     @varArray

varArray   = New(Array,1)     ← Memory Allocation
objName    = "ItemGroup"     ← Assignments
branchName = "ItemGroup"
key        = 'Stationaries'
varGroup   = VdsFindRecord(objName, VDS_EXCBLOBS, branchName, key)
           ← Identifying the Record to be modified

If(varGroup)
{
    varGroup.GroupName = key
    varGroup.MinValue  = 20

    ArrayAdd(varArray, varGroup)
    varRet = VdsModify(varArray, VDS_RECORD_TRAN) ← Modifying the Record
}
}
```

#### [Brief Description](#)

The above example will modify the record "Stationaries", in ItemGroup Object with MinValue as 20.

## Vaakya Keywords and Functions Reference Guide

### **VdsDelete**

#### ✓ [Function Brief](#)

Function to delete an existing record / reset / cluster from an object.

#### ✓ [Function Syntax](#)

Output = VdsDelete(ptrset, trantype)

#### ✓ [Function Arguments](#)

1. RecordSet
2. Transaction Type. One of the following
  - a. VDS\_RECORD\_TRAN  
To Delete a single Record.
  - b. VDS\_RECSET\_TRAN  
To Delete a set of Records.
  - c. VDS\_CLUSTER\_TRAN  
To Delete a record into a Master Slave Object

Function Returns 1, if the record is deleted, else returns 0.

#### ✓ [Example1 \(Deleting a Single Record into a Database\)](#)

```
ItemGroup  varGroup           ← Object Variable Declaration
Int        varRet            ← Variable Declarations

Text       @objName, @branchName, @key
Array      @varArray

varArray   = New(Array,1)     ← Memory Allocation
objName    = "ItemGroup"     ← Assignments
branchName = "ItemGroup"
key        = 'Stationaries'

varGroup   = VdsFindRecord(objName, VDS_EXCBLOBS, branchName, key)
                                     ← Identifying the Record to be deleted

If(varGroup)
{
    ArrayAdd(varArray, varGroup)

    varRet = VdsDelete(varArray, VDS_RECORD_TRAN) ← Deleting the Record
}
```

#### [Brief Description](#)

The above example will delete the record "Stationaries", from "ItemGroup" Object.

## Vaakya Keywords and Functions Reference Guide

### **VdsGetRecord**

✓ [Function Brief](#)

Function to Fetch a Record from a database table based on a Record Id.

✓ [Function Syntax](#)

Output = VdsGetRecord(objName, incBlob/excBlob, recId)

✓ [Function Arguments](#)

1. Object Name
2. BLOB Flag
  - a. VDS\_INCBLOBS – Include any BLOB Data associated with the Record
  - b. VDS\_EXCBLOBS - Exclude any BLOB data associated with the Record.
3. Record Id

✓ [Example](#)

```
ItemGroup  varGroup           ← Object Variable Declaration

varGroup   = VdsGetRecord("ItemGroup", VDS_EXCBLOBS, 1)
           ← Fetching Record "1" from "ItemGroup"

varGroup will contain the first record from ItemGroup Object.
```

### **VdsFindRecord**

✓ [Function Brief](#)

Function to Fetch a record from a database table based on a Key.

✓ [Function Syntax](#)

Output = VdsFindRecord(objName, incBlob/excBlob, branchName, InputKey)

✓ [Function Arguments](#)

1. Object Name
2. BLOB
  1. VDS\_INCBLOBS – Include any BLOB Data associated with the Record
  2. VDS\_EXCBLOBS - Exclude any BLOB data associated with the Record.
3. Index name of the object.
4. Value based on which Find Record should be performed

✓ [Example](#)

```
ItemGroup  varGroup           ← Object Variable Declaration
Text       @GroupName, @keyName ← Variable Declaration

GroupName  = "Stationaries"    ← Assignments
Keyname    = "ItemGroup"

varGroup   = VdsFindRecord("ItemGroup", VDS_EXCBLOBS, keyname, GroupName)
           ← Fetching Record from "ItemGroup"
```

## Vaakya Keywords and Functions Reference Guide

### **VdsGetTable**

#### ✓ [Function Brief](#)

Function to get all or a set of records from an object based on a condition.

#### ✓ [Function Syntax](#)

Output= VdsGetTable(objName, incBlob/excBlob, Expression, recPerPage, uidCursor)

#### ✓ [Function Arguments](#)

1. Object Name
2. BLOB
  1. VDS\_INCBLOBS – Include any BLOB Data associated with the Record
  2. VDS\_EXCBLOBS - Exclude any BLOB data associated with the Record.
3. Expression for handling user defined Filter condition.
4. recPerPage – Number of records to be fetched at a time. If this is defined as "0" all records from the table will be fetched.
5. UidCursor – Returns the cursor position

#### ✓ [Example](#)

```
ItemGroup Recset @setGroup           ← Recset Declaration
Text      @varExpression, @uidCursor ← Variable Declaration

Int      recPerPage
recPerPage = 0

setGroup = VdsGetTable("ItemGroup",VDS_EXCBLOBS,varExpression,recPerPage,
uidCursor)
           ← Assigning Records from "ItemGroup" into RecSet

VdsFetchClose(uidCursor)
```

### **VdsGetRecordCount**

#### ✓ [Function Brief](#)

Function to get number of records in an Object.

#### ✓ [Function Syntax](#)

Output = VdsGetRecordCount(objectName)

#### ✓ [Function Arguments](#)

1. Object Name

#### ✓ [Example](#)

```
Int      varCount           ← Variable Declaration
varCount = VdsGetRecordCount("ItemGroup") ← Returns number of records
```

## Vaakya Keywords and Functions Reference Guide

### **VdsGetCluster**

#### ✓ [Function Brief](#)

Function to fetch a cluster record based on a Cluster Id. The return value will be a pointer set which contains master and related grid object records.

#### ✓ [Function Syntax](#)

```
Output = VdsGetCluster(objName, incBlob/excBlob, clusterId)
```

#### ✓ [Function Arguments](#)

1. Object Name
2. BLOB
  1. VDS\_INCBLOBS – To include any BLOB Data associated with the Record
  2. VDS\_EXCBLOBS - To exclude any BLOB data associated with the Record.
3. Cluster Id

#### ✓ [Example](#)

```
Bool          ret          ← Variable Declarations
Text          @objName
Int           recid, arrCnt, RetType, recsetCnt

Array         @vaSale
SaleBill     objSale      ← Object Variable Declarations
SaleBillDet  objSD, Recset @setSD

objName = "SaleBill"      ← Assignments
recid   = 1
vaSale  = VdsGetCluster(objName, VDS_EXCBLOBS, recid)
          ← Fetch Cluster based on Cluster Id

arrCnt = ArrayCount(vaSale)

If(arrCnt)
{
    objSale = ArrayGet(vaSale, 1, RetType)

If(objSale)
{
    ConsolePrint(objSale.BillNo)
    ConsolePrint(objSale.Date)
    ConsolePrint(objSale.Total)
}

setSD      = ArrayGet(vaSale, 2, RetType)
recsetCnt = RecsetCount(setSD)
ConsolePrint(recsetCnt)
```

## Vaakya Keywords and Functions Reference Guide

```
If(recsetCnt)
{
    Loop(setSD, objSD)
    {
        ConsolePrint(objSD.ItemName)
        ConsolePrint(objSD.Qty)
        ConsolePrint(objSD.Rate)
        ConsolePrint(objSD.Amount)
    }
}
Else
{
    ConsolePrint("No record found.")
}
```

### **VdsFindCluster**

#### ✓ [Function Brief](#)

Function to fetch a cluster record based on a Key. The return value will be a pointer set which contains master and related grid object records.

#### ✓ [Function Syntax](#)

Output = VdsFindCluster(objName, incBlob/excBlob, branchName, key)

branchName will be the key name of the master object.

Key will be the value based on this value the Find Record should be performed.

VDS\_INCBLOBS will include any BLOB data associated with the Records.

VDS\_EXCBLOBS will exclude any BLOB data associated with the Records.

#### ✓ [Example](#)

```
Bool        ret
Text        @objName, @branchName, @key
Int         arrCnt, RetType, recsetCnt
Array       @vaSale
```

```
SaleBill    objSale
SaleBillDet objSD, Recset @setSD
```

```
objName     = "SaleBill"
branchName  = "SaleBill"
key         = "BILL0001"
```

```
vaSale = VdsFindCluster(objName, VDS_EXCBLOBS, branchName, key)
arrCnt = ArrayCount(vaSale)
```

```
If(arrCnt)
{
    objSale = ArrayGet(vaSale, 1, RetType)

    If(objSale)
    {
        ConsolePrint(objSale.BillNo)
        ConsolePrint(objSale.Date)
        ConsolePrint(objSale.Total)
    }
}
```

## Vaakya Keywords and Functions Reference Guide

```
setSD = ArrayGet(vaSale, 2, RetType)

recsetCnt = RecsetCount(setSD)
ConsolePrint(recsetCnt)

If(recsetCnt)
{
    Loop(setSD, objSD)
    {
        ConsolePrint(objSD.ItemName)
        ConsolePrint(objSD.Qty)
        ConsolePrint(objSD.Rate)
        ConsolePrint(objSD.Amount)
    }
}
Else
{
    ConsolePrint("No record found.")
}
```

### **VdsFetchMore**

✓ [Function Brief](#)

Function to fetch a set of records based on the uid cursor value.

✓ [Function Syntax](#)

Output = VdsFetchMore(uidCursor, Direction)

✓ [Function Arguments](#)

1. Cursor Value
2. Fetch Direction
  - a. VDS\_FETCH\_NEXT – Fetch Next Set of Records.
  - b. VDS\_FETCH\_PREV – Fetch Previous Set of Records.

### **VdsFetchClose**

✓ [Function Brief](#)

Keyword to Complete the Fetch Function

✓ [Function Syntax](#)

Output =VdsFetchClose(uidCursor)

## Vaakya Keywords and Functions Reference Guide

### Entity Functions

#### **VdsAddEntity**

✓ [Function Brief](#)

Function to add a new Entity object into the 'Entity folder' of an Application Domain.

✓ [Function Syntax](#)

Output = VdsAddEntity(EntityName)

✓ [Function Argument](#)

1. Entity Name

✓ [Example](#)

```
Bool    varRet                               ← Variable Declarations
Text    @varEntity

varEntity = Concat("NewEntity")             ← Assignment
varRet    = VdsAddEntity(varEntity)         ← Adding a New Entity
```

#### **VdsClearEntity**

✓ [Function Brief](#)

Function to clear the contents of an Entity folder. This function will not remove the Entity Folder.

✓ [Function Syntax](#)

Output = VdsClearEntity(EntityName)

✓ [Function Argument](#)

1. Entity Name

✓ [Example](#)

```
Bool    varRet                               ← Variable Declarations
Text    @varEntity

varEntity = Concat("NewEntity")             ← Assignment
varRet    = VdsClearEntity(varEntity)       ← Clearing Contents of Entity Folder
```

#### **VdsDeleteEntity**

✓ [Function Brief](#)

Function to delete an Entity. This function will remove the Entity Folder.

✓ [Function Syntax](#)

Output = VdsDeleteEntity(EntityName)

✓ [Function Argument](#)

1. Entity Name

✓ [Example](#)

```
Bool    varRet                               ← Variable Declarations
Text    @varEntity
```

## Vaakya Keywords and Functions Reference Guide

```
varEntity = Concat("NewEntity")    ← Assignment
varRet    = VdsDeletEntity(varEntity)
```

### **VdsSaveEntityTran**

#### ✓ [Function Brief](#)

Function to save transaction data into an Entity Object.

#### ✓ [Function Syntax](#)

Output = VdsSaveEntityTran(EntityName, ptrset,TranType)

#### ✓ [Function Arguments](#)

1. Entity Name
2. Pointer Variable
3. Transaction Type. Could be one of the following
  - a. VDS\_RECORD\_TRAN  
To Save a single Record.
  - b. VDS\_RECSET\_TRAN  
To Save a set of Records.
  - c. VDS\_CLUSTER\_TRAN  
To Save a Master Slave Record

#### ✓ [Example1 \(Adding a Single Record into an Entity\)](#)

```
Bool    varRet
Text    @varEntity
```

```
varEntity = Concat("NewEntity")
varRet    = VdsAddEntity(varEntity)
```

```
Text        @uidEntityName    ← Variable Declaration
Int         varRet
Array       @varArray
```

```
ItemGroup  varGroup          ← Object Variable Declaration
```

```
varArray   = New(Array,1)     ← Memory Allocation
varGroup   = New(ItemGroup, 1)
```

```
uidEntityName      = "20102011"    ← Assignment
varGroup.GroupName = 'Stationaries'
varGroup.MinValue  = 10
```

```
ArrayAdd(varArray, varGroup)
varRet = VdsSaveEntityTran(uidEntityName, varArray, VDS_RECORD_TRAN)
        ← Saves a record into the Entity Object
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example 2 \(Adding a Set of Records into an Entity\)](#)

```
ItemGroup  varGroup, Recset @setGroup      ← Object Variable Declaration
Int        varRet,i                       ← Variable Declaration
Text      @varText, @uidEntityName
Array     @varArray

varArray   = New(Array,1)                  ← Memory Allocation
varGroup   = New(ItemGroup, 1)
setGroup   = NewRecset(ItemGroup, 1)

I          = 0
uidEntityName = "20102011"                ← Assignment

While(i<= 2)
{
  varText          = Concat("Group",i)     ← Assignments
  varGroup.GroupName = varText
  varGroup.MinValue = 10
  RecsetAppend(setGroup, varGroup)

  i+=1
}
ArrayAdd(varArray, setGroup)
varRet = VdsSaveEntityTran(uidEntityName, varArray, VDS_RECSET_TRAN)
      ← Saves 2 Records into the Entity Object
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example 3 \(Adding a Cluster Entity\)](#)

```
Int          varRet, total, i          ← Variable Declarations
Text        @varText, @date, @uidEntityName

SaleBill    varSale                    ← Object Variable Declarations
SaleBillDet Recset @setSales
SaleBillDet varSalesDet
Array       @varArray

varArray    = New(Array,1)             ← Memory Allocation
varSale     = New(SaleBill,1)
varSalesDet = New(SaleBillDet,1)
setSales    = NewRecset(SaleBillDet,1)

uidEntityName = "20102011"             ← Assignments
varSale.BillNo = "BILL0001"
date          = Concat("16", "-07-", "2010")
varSale.Date  = TextParseDate(date, ' ', 1)
ArrayAdd(varArray, varSale)

i = 1
While(i <= 2)
{
    varText          = Concat("ItemName",i)
    varSalesDet.ItemName = varText
    varSalesDet.Qty   = 10
    varSalesDet.Rate  = 10
    varSalesDet.Amount = (varSalesDet.Qty * varSalesDet.Rate)
    total            += varSalesDet.Amount
    RecsetAppend(setSales, varSalesDet)

    i+= 1
}
varSale.Total      = total
ArrayAdd(varArray, setSales)

varRet = VdsSaveEntityTran(uidEntityName, varArray, VDS_CLUSTER_TRAN)
      ← Saves a Record into a Cluster Entity Object
```

## Vaakya Keywords and Functions Reference Guide

### **VdsModifyEntityTran**

#### ✓ [Function Brief](#)

Function to modify an existing record / recset / cluster from an entity Object.

#### ✓ [Function Syntax](#)

Output = VdsModifyEntityTran(EntityName,ptrset, trantype)

#### ✓ [Function Arguments](#)

1. Entity Name
2. Pointer Variable
3. Transaction Type. Could be one of the following
  - a. VDS\_RECORD\_TRAN - To Modify a single Entity Record.
  - b. VDS\_RECSET\_TRAN - To modify a set of Records.
  - c. VDS\_CLUSTER\_TRAN - To Modify a Master Slave Record

#### ✓ [Example1 \(Modifying an Entity Record\)](#)

```
ItemGroup  varGroup          ← Object Variable Declaration
Int        varRet           ← Variable Declarations
Text      @objName, @branchName
Text      @key, @uidEntityName
Array     @varArray

varArray   = New(Array,1)    ← Memory Allocation
objName    = "ItemGroup"    ← Assignment
branchName = "ItemGroup"
key        = 'Stationaries'
uidEntityName = "20102011"

varGroup   = VdsFindEntityRecord(uidEntity, objName, VDS_EXCBLOBS,
branchName, key)           ← Fetching Entity Record based on a Key

If(varGroup)
{
  varGroup.GroupName = key
  varGroup.MinValue  = 20

  ArrayAdd(varArray, varGroup)

  varRet = VdsModifyEntityTran(uidEntityName, varArray, VDS_RECORD_TRAN)
  ← Modifying Entity Record
}
```

## Vaakya Keywords and Functions Reference Guide

### **VdsDeleteEntityTran**

#### ✓ [Function Brief](#)

Function to delete an existing record / reset / cluster from an entity Object.

#### ✓ [Function Syntax](#)

Output = VdsDeleteEntityTran(EntityName, ptrset, trantype)

#### ✓ [Function Arguments](#)

1. Entity Name
2. Pointer Variable
3. Transaction Type. Could be one of the following
  - a. VDS\_RECORD\_TRAN - To Delete a single Entity Record.
  - b. VDS\_RECSET\_TRAN - To Delete a set of Records.
  - c. VDS\_CLUSTER\_TRAN - To Delete a Master Slave Record

#### ✓ [Example1 \(Deleteing an Entity Record\)](#)

```
ItemGroup  varGroup           ← Object Variable Declarations
Int        varRet             ← Variable Declarations
Text      @objName, @branchName
Text      @key, @uidEntityName
Array     @varArray
varArray  = New(Array,1)     ← Memory Allocation

objName    = "ItemGroup"     ← Assignments
branchName = "ItemGroup"
key        = 'Stationaries'
uidEntityName = "20102011"

varGroup = VdsFindEntityRecord(uidEntity,objName,VDS_EXCBLOBS,branchName, key)
          ← Fetching Entity Record based on a Key

If(varGroup)
{
  ConsolePrint(varGroup.RecId)
  ArrayAdd(varArray, varGroup)

  varRet = VdsDeleteEntityTran(uidEntity, varArray, VDS_RECORD_TRAN)
          ← Deleting Entity Record
}
```

### **VdsGetEntityRecord**

#### ✓ [Function Brief](#)

Function to Fetch a record from an entity object based on the Record Id.

#### ✓ [Function Syntax](#)

Output = VdsGetEntityRecord(EntityName,objName, incBlob/excBlob, recId)

#### ✓ [Example](#)

```
ItemGroup  varGroup           ← Object Variable Declaration
Text      @uidEntityName     ← Variable Declarations
Text      @objName
Int        varRecId
```

## Vaakya Keywords and Functions Reference Guide

```
uidEntity = "20102011"           ← Assignments
objName   = "ItemGroup"
varRecId  = 1
varGroup  = VdsGetEntityRecord(uidEntity, objName, VDS_EXCBLOBS, varRecId)
           ← Fetching Entity Record based on a Record Id
```

### **VdsFindEntityRecord**

#### ✓ [Function Brief](#)

Function to Fetch a record from an entity object based on an Index Key.

#### ✓ [Function Syntax](#)

Output = VdsFindEntityRecord(EntityName,objName,IncBlob/excBlob, branchName, InputKey)

#### ✓ [Function Arguments](#)

1. Entity Name
2. Object Name
3. BLOB
  1. VDS\_INCBLOBS - To include any BLOB Data associated with the Record
  2. VDS\_EXCBLOBS - To exclude any BLOB data associated with the Record.
4. Index Name
5. Value based on which Find Record should be performed.

#### ✓ [Example](#)

```
ItemGroup varGroup           ← Object Variable Declaration
Text      @GroupName, @keyname ← Variable Declarations
Text      @uidEntityName
Text      @objName

GroupName      = "Stationaries" ← Assignments
keyname       = "ItemGroup"
objName       = "ItemGroup"
uidEntityName = "20102011"

varGroup = VdsFindEntityRecord(uidEntityName,objName,VDS_EXCBLOBS,keyname, GroupName)
           ← Fetching Entity Record based on a Key

ConsolePrint(varGroup.MinValue)
```

## Vaakya Keywords and Functions Reference Guide

### **VdsGetEntityTable**

#### ✓ [Function Brief](#)

Function to get all or a set of records from an entity object based on condition.

#### ✓ [Function Syntax](#)

```
Output = VdsGetEntityTable(EntityName, objName, incBlob/excBlob, Expression,
recPerPage, uidCursor)
```

#### ✓ [Function Arguments](#)

1. Entity Name
2. Object Name
3. BLOB
  1. VDS\_INCBLOBS - To include any BLOB Data associated with the Record.
  2. VDS\_EXCBLOBS - To exclude any BLOB data associated with the Record.
4. Expression for handling user defined Filter condition.  
Ex: ItemGroup > 10. Only fields with structured data type can be used.
5. recPerPage – Number of records to be fetched at a time. If this is defined as "0" all records from the table will be fetched.
6. Cursor position.

#### ✓ [Example](#)

```
ItemGroup Recset @setGroup           ← Recset Declaration
Text      @varExpression, @uidCursor ← Variable Declarations
Text      @uidEntityName, @objName
Int       recPerPage , count

recPerPage = 0           ← Assignments
uidEntity  = "20102011"
objName    = "ItemGroup

setGroup= VdsGetEntityTable(uidEntityName,objName,VDS_EXCBLOBS,varExpression,
recPerPage, uidCursor)
           ← Fetching Entity Records based on an Expression

VdsFetchClose(uidCursor)
```

## Vaakya Keywords and Functions Reference Guide

### **VdsGetEntityRecordCount**

✓ [Function Brief](#)

Function to get a count of records in an Entity Object

✓ [Function Syntax](#)

Output =VdsGetEntityRecordCount(EntityName,objectName)

✓ [Function Arguments](#)

1. Entity Name
2. Object Name

✓ [Example](#)

```
Int          varRecCount          ← Variable Declarations
Text         @uidEntityName, @objName

uidEntityName = "20102011"        ← Assignments
objName       = "ItemGroup"
varRecCount   = VdsGetEntityRecordCount(uidEntityName, objName)
ConsolePrint(varRecCount)
```

### **VdsGetEntityCluster**

✓ [Function Brief](#)

Function to fetch a cluster record from an Entity Object based on a Cluster Id. The return value will be a pointer set which contains master and related grid object records.

✓ [Function Syntax](#)

Output = VdsGetEntityCluster(EntityName, objName, incBlob/excBlob, clusterId)

✓ [Function Arguments](#)

1. Entity Name
2. Object Name
3. BLOB
  - a. VDS\_INCBLOBS – To include any BLOB Data associated with the Record
  - b. VDS\_EXCBLOBS - To exclude any BLOB data associated with the Record.
4. Cluster Id

✓ [Example](#)

```
Bool          varRet              ← Variable Declarations
Text         @objName, @uidEntityName
Int          recid, arrCnt, RetType, recsetCnt
Array        @vaSale

SaleBill     objSale              ← Object Variable Declarations
SaleBillDet  objSD, Recset @setSD

objName      = "SaleBill"         ← Assignments
recid        = 1
uidEntityName = "20102011"

vaSale = VdsGetEntityCluster(uidEntityName, objName, VDS_EXCBLOBS, recid)
← Fetching Cluster Record from an Entity Object based on a Record Id
```

## Vaakya Keywords and Functions Reference Guide

```
arrCnt = ArrayCount(vaSale)

If(arrCnt)
{
    objSale = ArrayGet(vaSale, 1, RetType)

    If(objSale)
    {
        ConsolePrint(objSale.BillNo)
        ConsolePrint(objSale.Date)
        ConsolePrint(objSale.Total)
    }

    setSD      = ArrayGet(vaSale, 2, RetType)
    recsetCnt = RecsetCount(setSD)

    ConsolePrint(recsetCnt)

    If(recsetCnt)
    {
        Loop(setSD, objSD)
        {
            ConsolePrint(objSD.ItemName)
            ConsolePrint(objSD.Qty)
            ConsolePrint(objSD.Rate)
            ConsolePrint(objSD.Amount)
        }
    }
}
Else
{
    ConsolePrint("No record found.")
}
```

### **VdsFindEntityCluster**

#### ✓ [Function Brief](#)

Function to fetch a cluster record based on a Key from an Entity Object. The return value will be a pointer set which contains master and related grid object records.

#### ✓ [Function Syntax](#)

Output=VdsFindEntityCluster(EntityName,objName,incBlob/excBlob,branchName,key)

#### ✓ [Example](#)

```
Bool      ret
Text      @objName, @branchName, @key, @uidEntity
```

```
Int      arrCnt, RetType, recsetCnt
Array    @vaSale
```

```
SaleBill  objSale
SaleBillDet objSD, Recset @setSD
```

```
objName    = "SaleBill"
branchName = "SaleBill"
key        = "BILL0001"
```

```
uidEntity = "20102011"
vaSale=VdsFindEntityCluster(uidEntity,objName,VDS_EXCBLOBS,branchName,key)
```

## Vaakya Keywords and Functions Reference Guide

```
arrCnt = ArrayCount(vaSale)

If(arrCnt)
{
    objSale = ArrayGet(vaSale, 1, RetType)

    If(objSale)
    {
        ConsolePrint(objSale.BillNo)
        ConsolePrint(objSale.Date)
        ConsolePrint(objSale.Total)
    }

    setSD = ArrayGet(vaSale, 2, RetType)

    recsetCnt = RecsetCount(setSD)
    ConsolePrint(recsetCnt)

    If(recsetCnt)
    {
        Loop(setSD, objSD)
        {
            ConsolePrint(objSD.ItemName)
            ConsolePrint(objSD.Qty)
            ConsolePrint(objSD.Rate)
            ConsolePrint(objSD.Amount)
        }
    }
}
Else
{
    ConsolePrint("No record found.")
}
```

## Vaakya Keywords and Functions Reference Guide

### JTV Functions

#### ✓ [Function Brief](#)

JIT (Just in Time) Transaction Validator is a unique Vaakya Proc function similar to OLTP. JIT can be used to check current record status of an Object in real time, before committing a transaction into the database.

JIT is applicable for application Objects where concurrent requests are validated just before committing into the database.

### **JtvFindRecord**

#### ✓ [Function Brief](#)

JIT can be used to check current record status of an Object in real time, before committing. JtvFindRecord is the only keyword allowed in JIT procs. It will find a record from an object based on key.

#### ✓ [Function Syntax](#)

Output = JtvFindRecord(objName, incBlob/excBlob, branchName, InputKey)

#### ✓ [Function Arguments](#)

1. Object Name
2. BLOB
  - a. VDS\_INCBLOBS – To include any BLOB Data associated with the Record.
  - b. VDS\_EXCBLOBS - To exclude any BLOB data associated with the Record.
3. Index Name to be used for Finding the record
4. Value based on which the Record will be fetched.

#### ✓ [Example](#)

```
ItemGroup  varGroup, varGroup1           ← Object Variable Declaration
Text       @GroupName, @objName         ← Variable Declaration

GroupName = "Stationaries"              ← Assignments
objName   = "ItemGroup"

varGroup   = JtvFindRecord(objName, VDS_EXCBLOBS, keyname , GroupName)
```

## Vaakya Keywords and Functions Reference Guide

### Recset Functions

#### RecsetAppend

##### ✓ [Function Brief](#)

Function to append record(s) in to a temporary set. Function Returns 1 if records are appended else returns 0.

##### ✓ [Function Syntax](#)

Output = RecsetAppend(Recset, Recset/ Recptr)

##### ✓ [Function Arguments](#)

1. Record Set
2. Record Set Pointer Variable

##### ✓ [Example](#)

```
Int      varRet, i           ← Variable Declarations
Text     varText
Item     varItem           ← Object Variable Declaration
Item     Recset @setItem    ← Recset Declaration

setItem = NewRecset(Item,1) ← Memory Allocation
varItem = New(Item, 1)

i = 0

While(i<= 2)
{
    varText          = Concat("Group",i)
    varGroup.GroupName = varText          ← Assignment
    varGroup.MinValue = '10'
    varRet = RecsetAppend(setGroup, varGroup) ← Append values to the Recset

    i+=1
}
```

#### **Note:**

- If the second argument is a recset, the same object variable should be used.

## Vaakya Keywords and Functions Reference Guide

### **RecsetCount**

✓ [Function Brief](#)

Function to count number of records in the Record Set.

✓ [Function Syntax](#)

Output = RecsetCount(Recset)

✓ [Function Argument](#)

1. Record Set

✓ [Example](#)

```
Int      i, varCount      ← Variable Declarations
Text     varText
Item     varItem         ← Object Variable Declaration
Item     Recset @setItem  ← Recset Declaration

setItem = NewRecset(Item,1) ← Memory Allocation
varItem = New(Item, 1)

i = 0
While(i<= 2)
{
    varText          = Concat("Group",i) ← Assignments
    varGroup.GroupName = varText
    varGroup.MinValue = '10'

    RecsetAppend(setGroup, varGroup) ← Appending Values to Recset
    i+=1
}

varCount = RecsetCount(setItem) ← Function to Count Records in Set
```

### **RecsetFetch**

✓ [Function Brief](#)

Function to fetch a Record from a set corresponding to a record Index (similar to ROW ID in SQL).

✓ [Function Syntax](#)

Output = RecsetFetch(Recset, Index)

✓ [Function Arguments](#)

1. Record Set  
2. Record's Index Number

✓ [Example](#)

```
Text     uidCursor, @objName ← Variable Declarations
Int      recPerPage

Item     varItem         ← Object Variable Declaration
Item     Recset @setItem  ← Recset Declaration

objName = Concat("Item") ← Assignment
```

## Vaakya Keywords and Functions Reference Guide

```
setItem = dsGetTable(objName,VDS_EXCBLOBS,varExpression,recPerpage,uidCursor)
        ← Assigning contents of an object to a Record Set

varItem = RecsetFetch(setItem, 1)
        ← Fetching 1st Record from the Record Set
```

varItem will contain the first record of the recset.

### **RecsetFind**

#### ✓ [Function Brief](#)

Function to find a specific record from a Set of records.

#### ✓ [Function Syntax](#)

Output = RecsetFind(Recset, Recptr, (condition))

#### ✓ [Function Arguments](#)

1. Record Set
2. Record Set Pointer Variable
3. User defined Condition ( If Any)

#### ✓ [Example](#)

```
Text    uidCursor, @objName    ← Variable Declarations
Int     recPerPage
```

```
Item    varItem, varItem1     ← Object Variable Declaration
Item    Recset @setItem       ← Recset Declaration
```

```
objName = Concat("Item")      ← Assignment
```

```
setItem = VdsGetTable(objName,VDS_EXCBLOBS,varExpression,recPerpage,uidCursor)
        ← Assigning contents of an object to a Record Set
```

```
varItem1 = RecsetFind(setItem, varItem, (varItem.RecId == 2))
        ← Fetching 2nd Record from the Record Set
```

varItem1 will contain the record with RecId 2

## Vaakya Keywords and Functions Reference Guide

### **RecsetFilter**

✓ [Function Brief](#)

Function to remove records from a Recset that satisfy a specific condition.

✓ [Function Syntax](#)

```
Return = RecsetFilter(Recset, Recptr, (condition))
```

✓ [Function Arguments](#)

1. Record Set
2. Record Set Pointer Variable
3. User defined Filter Condition

✓ [Example](#)

```
Int      varRet, recPerpage      ← Variable Declarations
Text     uidCursor, @objName

Item     varItem                 ← Object Variable Declarations
Item     Recset @setItem         ← Recset Declaration

varItem = New(Item, 1)          ← Memory Allocation
objName = Concat("Item")       ← Assignment

setItem = VdsGetTable(objName, VDS_EXCBLOBS, varExpression, recPerpage, uidCursor)
                                                ← Assigning contents of an object to a Record Set

varRet = RecsetFilter(setItem, varItem, (varItem.RecId == 2))
                                                ← Defining Filter Function
```

Function will return 1 if Filter condition is successful else the function will return 0. The updated RecSet will have all records other that Record Id 2

### **RecsetSplit**

✓ [Function Brief](#)

Function to Split a Record Set based on a condition. Function returns a new set with records satisfying the split condition.

✓ [Function Syntax](#)

```
NewSet = RecsetSplit(Recset, Recptr, (condition))
```

✓ [Function Arguments](#)

1. Record Set
2. Record Set Pointer Variable
3. User defined Split Condition

✓ [Example](#)

```
Int      recPerpage              ← Variable Declarations
Text     uidCursor, @objName

Item     varItem                 ← Object Variable Declaration
Item     Recset @setItem, Recset @setItem1 ← Recset Declaration

objName = Concat("Item")       ← Assignment
```

## Vaakya Keywords and Functions Reference Guide

```
setItem=VdsGetTable(objName,VDS_EXCBLOBS,varExpression,recPerpage,uidCursor)
    ← Assigning contents of an object to a Record Set
```

```
setItem1 = RecsetSplit(setItem, varItem, (varItem.RecId > 3))
    ← Defining Filter to Split a Set
```

setItem1 will contain the records with Record Id greater than 3.

### **RecsetOrder**

#### ✓ [Function Brief](#)

Function to Sort the records in a Record Set. Sort could be in ascending or descending order. Please note that Sort can be applied only on a single field.

#### ✓ [Function Syntax](#)

```
Output = RecsetOrder(Recset ,fldName ,sortOrder)
```

#### ✓ [Function Arguments](#)

1. Record Set
2. Field on which Sort has to be applied.
3. Sort Order
  - a. ASC - Sort in Ascending Order
  - b. DSC – Sort in Descending Order.

#### ✓ [Example](#)

```
Int      varFilter, varRet, recPerpage      ← Variable Declarations
Text     uidCursor, @objName

Item     varItem                            ← Object Variable Declaration

Item     Recset @setItem, Recset @setItem1  ← VRecset Declarations

VarRet = 0                                  ← Assignments
objName= Concat("Item")

setItem= VdsGetTable(objName,VDS_EXCBLOBS,varExpression,recPerpage,uidCursor)
    ← Assigning contents of an object to a Record Set

varRet = RecsetOrder(setItem, Price,ASC)
    ← Sorting Price Field in Ascending Order

setItem will contain the records Ordered by Price in the Ascending Order.
```

### **RecsetAggregate**

#### ✓ [Function Brief](#)

Function to aggregate column(s) in a Recset.

#### ✓ [Function Syntax](#)

```
Output = RecsetAggregate(Recset , Recptr ,SUM(fldName)..)
```

#### ✓ [Function Arguments](#)

1. Record Set
2. Record Set Pointer Variable

## Vaakya Keywords and Functions Reference Guide

### 3. Aggregate Function

#### ✓ [Example](#)

```
Int      varAggregate, recPerpage ← Variable Declarations
Text     uidCursor, @objName
```

```
Item     varItem                  ← Object Variable Declarations
```

```
Item     Recset @setItem          ← Recset Declarations
```

```
Item     Recset @setItem1
```

```
objName = Concat("Item")         ← Assignments
```

```
setItem = VdsGetTable("Item",VDS_EXCBLOBS,varExpression,recPerpage,uidCursor)
          ← Assigning contents of an object to a Record Set
```

```
varAggregate = RecsetAggregate(setItem, varItem, SUM(Price))
          ← Set Aggregate Function on Price Column
```

setItem will contain the aggregated record.

## Vaakya Keywords and Functions Reference Guide

### **RecsetGroup**

✓ [Function Brief](#)

Function to be used in conjunction with aggregate functions to group the result set on one column (similar to group by clause in SQL)

✓ [Function Syntax](#)

```
Output = RecsetGroup(Recset, fldName, SUM(fldName)..)
```

✓ [Function Arguments](#)

1. Record Set
2. Field / Column Name
3. Aggregate Function to be applied.

✓ [Example](#)

```
Int      varGroup, recPerpage      ← Variable Declaration
Text     uidCursor, @objName
Item     varItem                   ← Object Variable Declaration

Item     Recset @setItem           ← Recset Declaration
Item     Recset @setItem1

objName = Concat("Item")          ← Assignment

setItem=VdsGetTable(objName,VDS_EXCBLOBS,varExpression,recPerpage, uidCursor)
        ← Assigning contents of an object to a Record Set

setItem1 = RecsetGroup(setItem, Price, COUNT(Id))
        ← Aggregate Function on Price Column

setItem1 will contain the grouped record
```

## Vaakya Keywords and Functions Reference Guide

### **RecsetDuplicate**

✓ [Function Brief](#)

Function to create a duplicate a Set with its contents.

✓ [Function Syntax](#)

Output = RecsetDuplicate(Recset)

✓ [Function Argument](#)

1. RecSet to be duplicated

✓ [Example](#)

```
Int      recPerpage           ← Variable Declaration
Text     uidCursor, @objName
Item     Recset @setItem, Recset @setItem1 ← Recset Declaration

objName = Concat("Item")      ← Assignment

setItem=VdsGetTable(objName,VDS_EXCBLOBS,varExpression,recPerpage,uidCursor)
        ← Assigning contents of an object to a Record Set

setItem1 = RecsetDuplicate(setItem)
        ← Creating a duplicate Set

setItem1 will contain the duplicated Records.
```

## Array Functions

### **ArrayAdd**

✓ [Function Brief](#)

Function to add a Set of Values into an Array.

✓ [Function Syntax](#)

Output = ArrayAdd(ptrSet, ptr)

✓ [Function Arguments](#)

1. Array Pointer Variable
2. Any Pointer Variable

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
Bool          varRet          ← Variable Declaration
Array        @varArray
ItemGroup    varGroup        ← Object Variable Declaration

varArray = New(Array, 1)      ← Memory Allocation
varGroup = New(ItemGroup, 1)

varGroup.GroupName = "Test Group" ← Assignment
varGroup.MinVal    = 10

varRet = ArrayAdd(varArray, varGroup)
          ← Adding into Array
```

Returns 1 if values are added into the Array.

## **ArrayGet**

### ✓ [Function Brief](#)

Function to get the record of a given index from an array.

### ✓ [Function Syntax](#)

```
Output = ArrayGet(ptrSet, slot, ptrToArg)
```

### ✓ [Function Arguments](#)

1. Array Pointer Variable
2. Slot Number
3. Pointer Type Variable

### ✓ [Example](#)

```
Array        @varArray          ← Variable Declarations
Int          varType

varGroup = ArrayGet(varArray, 1, varType)
          ← Array Get
```

varGroup will contain the values in the 1 slot.

varType will be the type of value should be returned.

## **ArrayGetLast**

### ✓ [Function Brief](#)

Function to get the values from the Last Index of an Array.

### ✓ [Function Syntax](#)

```
Output = ArrayGetLast(ptrSet, ptr/ptrtypearg)
```

### ✓ [Function Arguments](#)

1. Array Pointer Variable
2. Pointer Type Variable

### ✓ [Example](#)

```
Bool          varRet          ← Variable Declarations
```

## Vaakya Keywords and Functions Reference Guide

Array @varArray

ItemGroup varGroup ← Object Variable Declarations

```
varGroup = ArrayGetLast(varArray, varGroup)
           ← ArrayGetLast Function
```

varGroup will contain the values in the 1 slot.

### **ArrayCount**

#### ✓ [Function Brief](#)

Function to count number of records in an Array.

#### ✓ [Function Syntax](#)

```
Output = ArrayCount(ptrSet)
```

#### ✓ [Function Argument](#)

1. Array Pointer Variable

#### ✓ [Example](#)

```
ItemGroup varGroup           ← Object Variable Declaration
Int        varRet, varCount   ← Variable Declarations
Array      @varArray
```

```
varArray = New(Array, 1)      ← Memory Allocation
```

```
varGroup.GroupName = "Test Group" ← Assignments
varGroup.MinVal    = 10
```

```
varRet = ArrayAdd(varArray, varGroup)
varCount = ArrayCount(varArray) ← Gets a Count of Array
```

varCount will return 1.

## Vaakya Keywords and Functions Reference Guide

### Set Functions

#### SetAdd

✓ [Function Brief](#)

Function to add a set of Values into temporary set.

✓ [Function Syntax](#)

```
Output = SetAdd(Set, "value")
```

✓ [Function Arguments](#)

1. Set Name
2. Value to be Added

✓ [Example](#)

```
Int      varRet           ← Variable Declaration
Set      @varSet         ← Set Declaration
varSet = New(Set, 1)      ← Memory Allocation for Set

varRet = SetAdd(varSet, "test") ← Assigning Values to Set
```

Returns 1, if the value is added to the set.

#### SetDelete

✓ [Function Brief](#)

Function to Delete Values from a temporary set. **Please read this section in continuation to SetAdd Function.**

✓ [Function Syntax](#)

```
Output = SetDelete(Set, "value")
```

✓ [Function Arguments](#)

1. Set Name
2. Value to be Deleted

✓ [Example](#)

```
Int      varRet           ← Variable Declaration
Set      @varSet         ← Set Declaration
varRet = SetDelete(varSet, "test") ← Deleting Set Values
```

Returns 1 if the value is deleted from the set.

## Vaakya Keywords and Functions Reference Guide

### SetFetch

✓ [Function Brief](#)

Function to Fetch Values from a set, based on an index. **Please read this section in continuation to SetAdd Function.**

✓ [Function Syntax](#)

```
Output = SetFetch(Set, Index)
```

✓ [Function Arguments](#)

1. Set Name
2. Index Number to be Fetched

✓ [Example](#)

```
Set      @varSet      ← Set Declaration
Text     @varFetch    ← Variable Declaration

varFetch = SetFetch(varSet,1) ← SetFetch Based on an Index

varFetch will contain the first value of the set.
```

### SetFetchInOrder

✓ [Function Brief](#)

Function to fetch values from a set in ascending order based on an index. **Please read this section in continuation to SetAdd Function.**

✓ [Function Syntax](#)

```
Output = SetFetchInOrder(Set, Index)
```

✓ [Function Arguments](#)

1. Set Name
2. Index Number to be Fetched

✓ [Example](#)

```
Set      @varSet      ← Set Declaration
Text     @varFetch    ← Variable Declaration
varFetch = SetFetchInOrder(varSet,1) ← Fetch in Ascending Order

varFetch will contain the first value of the set in the ascending order.
```

## Vaakya Keywords and Functions Reference Guide

### **SetContains**

✓ [Function Brief](#)

Function to check whether a given value is available in a set. **Please read this section in continuation to SetAdd Function.**

✓ [Function Syntax](#)

```
Output = SetContains(Set, "value")
```

✓ [Function Arguments](#)

1. Set Name
2. Value to be Checked

✓ [Example](#)

```
Int      varRet      ← Variable Declaration  
Set      @varSet     ← Set Declaration
```

```
varRet = SetContains(varSet, "test") ← Function Set Contains
```

If found the function returns 1.

### **SetEqual**

✓ [Function Brief](#)

Function to check whether two sets are equal.

✓ [Function Syntax](#)

```
Output = SetEqual(Set1, Set2)
```

✓ [Function Arguments](#)

1. 1st Set
2. 2<sup>nd</sup> Set

✓ [Example](#)

```
Int      varRet      ← Variable Declaration  
Set      @varSet1, @varSet2 ← Set Declaration
```

```
varRet = SetEqual(varSet1, varSet2) ← Function SetEqual
```

Function Returns 1 if both sets are equal.

## Vaakya Keywords and Functions Reference Guide

### SetCount

✓ [Function Brief](#)

SetCount returns the number of values in a set. **Please read this section in continuation to SetAdd Function.**

✓ [Function Syntax](#)

Output = SetCount(Set)

✓ [Function Arguments](#)

1. SetName

✓ [Example](#)

```
Int      varCount      ← Variable Declaration
Set      @varSet       ← Set Declaration
varCount = SetCount(varSet) ← Function SetCount
```

Function returns number of records in a Set.

### SetJoin

✓ [Function Brief](#)

Function to form a new Set by joining two different Sets. The resultant set will have all values from the two Sets.

✓ [Function Syntax](#)

Output = SetJoin(Set1, Set2)

✓ [Function Arguments](#)

1. 1<sup>st</sup> Set  
2. 2<sup>nd</sup> Set

✓ [Example](#)

```
Int      varRet      ← Variable Declaration
Set      @varSet1,@varset2, @varSet3 ← Set Declaration
```

```
varSet1 = {1,2,3} ← Set Assignments
varSet2 = {4,5,6}
varSet3 = SetJoin(varSet1, varSet2) ← Set Join Function
```

varSet3 will contain {1,2,3,4,5,6}.

### SetUnion

✓ [Function Brief](#)

Function to create a New Set, which has members from "Set A" & "Set B".

✓ [Function Syntax](#)

Output = SetUnion(Set1, Set2)

✓ [Function Arguments](#)

1. 1<sup>st</sup> Set

## Vaakya Keywords and Functions Reference Guide

### 1. 2<sup>nd</sup> Set

#### ✓ [Example](#)

```
Int      varRet          ← Variable Declaration
Set      @varSet1, @varset2, @varSet3 ← Set Declarations

varSet1 = {1,2,3}      ← Set Assignments
varSet2 = {2,3,4}

varSet3= SetUnion(varSet1, varSet2) ← Joining Two Sets

varSet3 will contain{1,2,3,4}.
```

## SetIntersect

#### ✓ [Function Brief](#)

Function to create a New Set, that contains all elements Of "Set A" that also belong to "Set B" or vice versa.

#### ✓ [Function Syntax](#)

```
Output = SetIntersect(Set1, Set2)
```

#### ✓ [Function Arguments](#)

1. 1<sup>st</sup> Set
2. 2<sup>nd</sup> Set

#### ✓ [Example](#)

```
Int      varRet          ← Variable Declaration
Set      @varSet1,@varset2, @varSet3 ← Set Declarations

varSet1 = {1,2,3}      ← Set Assignments
varSet2 = {2,3,4}

varSet3= SetIntersect(varSet1, varSet2) ← Set Intersection

varSet3 will contain{2,3}.
```

## Dict Functions

### DictAdd

#### ✓ [Function Brief](#)

Function used to create a new Dictionary. "Dictionary" is a Vaakya Data Type, for storing Key Value pairs.

#### ✓ [Function Syntax](#)

```
Output = DictAdd(Dict, Key, value)
```

#### ✓ [Function Arguments](#)

1. Dictionary
2. Dictionary Key. This should be a unique name.
3. Values to be stored into the Dictionary.

#### ✓ [Example](#)

```
Int      varRet          ← Variable Declaration
```

## Vaakya Keywords and Functions Reference Guide

```
Dict      @glbDict      ← Dictionary Declaration
glbDict = New(Dict, 1)  ← Memory Allocation

varRet = DictAdd(glbDict, "name", "vaakya")
                                     ← Adding Values to Dictionary
```

Function Returns 1 Dictionary is Added.

### **DictFind**

#### ✓ [Function Brief](#)

Function to find values from a Dictionary.

#### ✓ [Function Syntax](#)

```
Output = DictFind(dict, Key)
```

#### ✓ [Function Arguments](#)

1. Dictionary Name
2. Dictionary Key.

#### ✓ [Example](#)

```
Text      @dictstr      ← Variable Declarations
Dict      @glbDict      ← Dictionary Declaration
glbDict = New(Dict, 1)  ← Memory Allocation

varRet = DictAdd(glbDict, "name", "vaakya")
                                     ← Adding Values to Dictionary

If(varRet)
{
    dictstr = DictFind(glbDict, "name")
                                     ← Finding Values from Dictionary
}
dictstr will contain "vaakya".
```

### **DictFindNext**

#### ✓ [Function Brief](#)

Function to find multiple values (If any) from a Dictionary.

#### ✓ [Function Syntax](#)

```
Output = DictFindNext(dict, Key)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Arguments](#)

1. Dictionary Name
2. Dictionary Key

### ✓ [Example](#)

```
Int      i                ← Variable Declarations
Text     @dictstr, @ptrAdd
Dict     @glbDict        ← Dictionary Declaration
glbDict = New(Dict, 1)   ← Memory Allocation

i = 0
While(i <= 3)
{
    ptrAdd = Concat("name", i)
    DictAdd(glbDict, "name", ptrAdd)
                                ← Adding Values to Dictionary
}

While(i <=3)
{
    dictstr = DictFindNext(glbDict, "name")
                                ← Fetching Dict Values
    i+=1
}
```

dictstr will return the values one by one.

## **DictReplace**

### ✓ [Function Brief](#)

Function to Replace a Dictionary Value.

### ✓ [Function Syntax](#)

```
Output = DictReplace(dict, Key, NewTextVal)
```

### ✓ [Function Arguments](#)

1. Dictionary Name
2. Dictionary Key
3. Replacement Value

### ✓ [Example](#)

```
Text     @dictstr        ← Variable Declarations
Bool     varRet
Dict     @glbDict        ← Dictionary Declaration
glbDict = New(Dict, 1)   ← Memory Allocation

varRet = DictAdd(glbDict, "name", "vaakya")
                                ← Adding Values to Dictionary

varRet = DictReplace(glbDict, "name", "test")
                                ← Replacing Dictionary Values
```

## Vaakya Keywords and Functions Reference Guide

```
If(varRet)
{
    dictstr = DictFind(glbDict, "name")
}
```

dictstr contains "test".

### DictCount

#### ✓ [Function Brief](#)

Function to get a count of Dictionary Key Value pairs.

#### ✓ [Function Syntax](#)

```
Output = DictCount(dict)
```

#### ✓ [Function Argument](#)

1. Dictionary Name

#### ✓ [Example](#)

```
Int        varRet, varCount    ← Variable Declarations
Dict       @glbDict           ← Dictionary Declaration
glbDict = New(Dict, 1)         ← Memory Allocation
```

```
varRet = DictAdd(glbDict, "name", "vaakya")
        ← Adding Values to Dictionary
```

```
If(varRet)
{
    varCount = DictCount(glbDict)
                ← Getting Count of Values in Dictionary
}
varCount will return 1.
```

## Stream Functions

### StreamOpen

#### ✓ [Function Brief](#)

Function to Open a Stream & allocate memory for a Stream variable.

#### ✓ [Function Syntax](#)

```
Output = StreamOpen(TypeStream)
```

#### ✓ [Function Argument](#)

- Argument 1 - Stream Type which could be any one of the three below
  1. STD\_STREAM
  2. LINE\_STREAM
  3. XML\_STREAM

#### ✓ [Example](#)

```
Stream      @varStream        ← Stream Declaration
varStream = StreamOpen(STD_STREAM) ← Opening a Standard Stream
```

## Vaakya Keywords and Functions Reference Guide

### StreamPrint

✓ [Function Brief](#)

Function to print value(s) into a Stream variable

✓ [Function Syntax](#)

```
Output = StreamPrint(Stream,Content1,Content2,...,Content n)
```

✓ [Function Arguments](#)

1. Stream Name
2. Contents to be printed into the Stream

✓ [Example](#)

```
Stream      @varStream      ← Stream Declaration  
Int         varRet          ← Variable Declaration
```

```
varStream = StreamOpen(STD_STREAM)  
           ← Opening a Standard Stream
```

```
varRet    = StreamPrint(varStream,"This is a example for stream print")  
           ← Printing into Stream
```

Function Returns 1 if StreamPrint is successful.

### StreamOpenFile

✓ [Function Brief](#)

Function to open a file, which will be used for printing values from a Stream.

✓ [Syntax](#)

```
Output = StreamOpenFile(FilePath,Type)
```

✓ [Function Arguments](#)

1. File Path – The function will create a file if it does not exist.
2. Stream Type - Please refer to "*StreamOpen*" function for Stream Types.

✓ [Example](#)

```
Stream      @varStream      ← Stream Declaration
```

```
varStream = StreamOpenFile("c:\StreamFile.txt", STD_STREAM)  
           ← Opening a File "StreamFile.txt" in C Drive
```

### StreamReadOpenFile

✓ [Function Brief](#)

Function Opens a file to read the contents and returns the contents of the file as a Stream.

✓ [Function Syntax](#)

```
Output = StreamReadOpenFile(FilePath, Type)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Arguments](#)

1. File Path
2. Stream Type - Please refer to "*StreamOpen*" function for Stream Types.

### ✓ [Example](#)

```
Stream      @varStream      ← Stream Declaration

varStream = StreamReadOpenFile("C:\StreamOpenFile.txt", STD_STREAM)
              ← Opening a File "StreamFile.txt" from C Drive

varStream will contain the contents of the file.
```

## **StreamInfo**

### ✓ [Function Brief](#)

Function to get the Stream size and size of the Stream Header.

### ✓ [Function Syntax](#)

```
Output = StreamInfo(stream, sizStreamPtr, sizHeaderPtr, Type)
```

### ✓ [Function Arguments](#)

1. Stream Name
2. Stream Size – Value for this will be filled by the function.
3. Stream Header Size – Value for this will be filled by the function.
4. Stream Type - Please refer to "*StreamOpen*" function for Stream Types.

### ✓ [Example](#)

```
Int          StreamSizePtr, HeaderSizePtr, varRet, typStream
              ← Variable Declarations

Stream      @varStream      ← Stream Declaration
varStream = StreamOpen(STD_STREAM) ← Opening a New Stream

StreamPrint(varStream, "This is an example") ← Printing into Stream

varRet= StreamInfo(varStream,StreamSizePtr,HeaderSizePtr,typStream)
              ← Getting Stream / Header Size

varRet will return 1 if StreamInfo is true.
  ✓ StreamSizePtr will return the Stream Size as 18.
  ✓ HeaderSizePtr will return the size of Stream Header.
```

## **StreamWrite**

### ✓ [Function Brief](#)

Function to write values into an existing Stream.

### ✓ [Function Syntax](#)

```
Output = StreamWrite(stream,streamPosition, writebuffer, writebuffersize)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Arguments](#)

1. Stream Name
2. Stream Position
3. Buffer to Write
4. Size of Buffer to be Written

### ✓ [Example](#)

```
Int      varRet, position, writesize    ← Variable Declarations
Text     @ptrWriteStr
```

```
Stream   @varStream                    ← Stream Declaration
varStream = StreamOpen(STD_STREAM)     ← Opening a New Stream
varRet   = StreamPrint(varStream, "This") ← Printing into Stream
ptrWriteStr = Concat("is")
```

```
position = 4
writesize = 2
varRet   = StreamWrite(varStream, position, ptrWriteStr, writesize)
                                     ← Writing into Stream
```

Function Returns 1 if StreamWrite is successful.

### **Note:**

- The position should be less than or equal to the StreamLength +1 .In the above example the position should not be more than 5.
- This function will overwrite the existing stream.

## **StreamAppend**

### ✓ [Function Brief](#)

Function to Append values into an existing Stream at the end.

### ✓ [Function Syntax](#)

```
Output = StreamAppend(stream, writebuffer, writebuffersize, streamsize)
```

### ✓ [Function Arguments](#)

1. Stream Name
2. Buffer to Write
3. Size of Buffer to be Written
4. Size of Stream

### ✓ [Example](#)

```
Int      varRet, varStreamsize, writesize ← Variable Declarations
Stream   @varStream                       ← Stream Declaration
```

```
varStream = StreamOpen(STD_STREAM)     ← Opening a New Stream
varRet   = StreamPrint(varStream, "This") ← Printing into Stream
writesize = 2
```

```
varRet   = StreamAppend(varStream, "is", writesize, varStreamsize)
                                     ← Appending into Stream
```

varRet will return 1 if the stream append operation is true.

## Vaakya Keywords and Functions Reference Guide

### **StreamRead**

✓ [Function Brief](#)

Function to Read through a stream from a specified position.

✓ [Function Syntax](#)

```
Output = StreamRead(stream, pos, buffer, buffersize)
```

✓ [Function Arguments](#)

1. Stream Name
2. Position from where Read has to Start
3. Read Stream will be stored into this buffer
4. Size of the buffer being Read.

✓ [Example](#)

```
Int          varRet, varStreamsize, position    ← Variable Declarations
Text         @varBuff
Stream       @varStream                       ← Stream Declaration
varStream    = StreamOpen(STD_STREAM)         ← Opening a New Stream
varRet       = StreamPrint(varStream, "This", -) ← Printing into Stream
varStreamsize = 4
position     = 1
varRet       = StreamRead(varStream, position, varBuff, varStreamsize)
              ← Reading from a Stream from Position 1

varRet will return 1 if the stream read operation is true.
```

### **StreamMapContent**

✓ [Function Brief](#)

Function to Map the contents of a Stream.

✓ [Function Syntax](#)

```
Output = StreamMapContent(stream, Startpos, sizetoread, sizeread)
```

✓ [Function Arguments](#)

1. Stream Name
2. Position from where Read has to Start
3. Size to be Read
4. Size of the buffer being Read

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
Int          varRet, size, position, header, typStream, sizeChunk
Text        @varBuff          ← Variable Declarations
Stream      @varStream        ← Stream Declaration

varStream = StreamOpen(STD_STREAM) ← Opening a New Stream

varRet     = StreamPrint(varStream, "To be Mapped")
                                     ← Printing into Stream

size       = 0
header     = 0
position   = 0
typStream  = 0
sizeChunk  = 0

varRet     = StreamInfo(varStream, size, header, typStream)
                                     ← Getting Stream / Header Size
varBuff    = StreamMapContent(varStream, position, size, sizeChunk)
```

## **StreamFind**

### ✓ [Function Brief](#)

Function to match a pattern in the Stream.

### ✓ [Function Syntax](#)

Output = StreamFind(stream, Startpos, Pattern, FoundPos)

### ✓ [Example](#)

```
Int          varRet, varFound, varStartPosition      ← Variable Declarations
Stream      @varStream                              ← Stream Declaration

varStream = StreamOpen(STD_STREAM)                  ← Opening a New Stream
varRet    = StreamPrint(varStream, "To be Found") ← Printing into Stream

varStartPosition = 0
varRet           = StreamFind(varStream, varStartPosition, "be", varFound)
                                     ← Finding Stream Content
```

Function will Return 1 if StreamFind is TRUE.

## **StreamScanBoundToken**

### ✓ [Function Brief](#)

Function to find a specific pattern with starting and ending position in the stream.

### ✓ [Function Syntax](#)

Output = StreamScanBoundToken(stream, begPattern, endPattern, posBegPtr, posEndPtr)

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Arguments](#)

1. Stream Name
2. Match Pattern Start
3. Match Pattern End
4. Starting Position
5. Ending Position

### ✓ [Example](#)

```
Int          varRet, varBegPos, varEndPos    ← Variable Declarations
Stream      @varStream                      ← Stream Declaration

varStream = StreamOpen(STD_STREAM)         ← Opening a New Stream

StreamPrint(varStream, "This is an example for checking ()")
                                                ← Printing into Stream

varRet = StreamScanBoundToken(varStream, "\",\"", varBegPos, varEndPos)
                                                ← Matching Patterns

varRet will return 1 if the specified pattern is found in the stream.
```

## **StreamCopy**

### ✓ [Function Brief](#)

Function to copy contents of a Stream to another based on the position and size.

### ✓ [Function Syntax](#)

```
Output = StreamCopy(Stream, pos, siz, destStream)
```

### ✓ [Function Arguments](#)

1. Stream Name
2. Starting Position to Copy contents
3. Size of contents to be copied.
4. Destination Stream

### ✓ [Example](#)

```
Int          varRet, position, size    ← Variable Declarations
Stream      @varStream, @varDest      ← Stream Declaration

varStream = StreamOpen(STD_STREAM) ← Opening a New Stream

varRet = StreamPrint(varStream, "To be copied",)
                                                ← Printing into Stream

position = 3
size = 2
varDest = StreamOpen(STD_STREAM) ← Opening a Destination Stream

varRet = StreamCopy(varStream, position, size, varDest)
                                                ← Copying into New Stream

varRet will return 1 if the stream is copied.
```

## Vaakya Keywords and Functions Reference Guide

### **StreamTruncate**

✓ [Function Brief](#)

Function to truncate a stream from a specified position.

✓ [Function Syntax](#)

Output = StreamTruncate(Stream, posTrunc)

✓ [Function Arguments](#)

1. Stream Name
2. Truncate Position

✓ [Example](#)

```
Int          varRet          ← Variable Declaration
Stream       @varStream      ← Stream Declarations

VarStream = StreamOpen(STD_STREAM) ← Opening a New Stream

varRet      = StreamPrint(varStream, "To be truncated")
                                     ← Printing into Stream

varRet      = StreamTruncate(varStream, 6)
                                     ← Truncating Stream
```

varRet will return 1 if the stream is truncated.

### **StreamSave**

✓ [Function Brief](#)

Function to save the contents of the stream to a file.

✓ [Function Syntax](#)

Output = StreamSave(Stream, FilePath)

✓ [Function Arguments](#)

1. Stream Name
2. File Path

✓ [Example](#)

```
Int          varRet          ← Variable Declarations
Text         @ptrPath        ← Stream Declaration
Stream       @varStream      ← Opening a New Stream
VarStream = StreamOpen(STD_STREAM)

varRet      = StreamPrint(varStream, "To be Copied",)
                                     ← Printing into Stream

ptrPath     = Concat("/VAAKYA/save.txt")

varRet      = StreamSave(varStream, ptrPath)
                                     ← Stream Save into a FilePath
```

varRet will return 1 if the stream is saved

## Vaakya Keywords and Functions Reference Guide

### **StreamClose**

✓ [Function Brief](#)

Function to close a stream.

✓ [Function Syntax](#)

```
Output = StreamClose(Stream)
```

✓ [Function Argument](#)

1. Stream Name

✓ [Example](#)

```
Int          varRet          ← Variable Declaration
```

```
Stream      @varStream      ← Stream Declaration
```

```
VarStream = StreamOpen(STD_STREAM) ← Opening a New Stream
```

```
varRet    = StreamClose(varStream) ← Closing a Stream
```

varRet will return 1 if the stream is closed.

## Vaakya Keywords and Functions Reference Guide

### **StreamFetchInit**

✓ [Function Brief](#)

Function to initialize a stream Fetch.

✓ [Function Syntax](#)

```
Output = StreamFetchInit(Stream, buffer)
```

✓ [Function Arguments](#)

1. Stream Name
2. Buffer Size

✓ [Example](#)

```
Bool      varRet          ← Variable Declarations
Int       varSize
Stream    @varStream      ← Stream Declaration
```

```
VarStream = StreamOpen(STD_STREAM) ← Opening a New Stream
varRet    = StreamFetchInit(varStream, varSize)
                                     ← Initialising Stream Fetch
```

varRet will return 1 if the StreamFetch is initialized.

### **StreamFetchNext**

✓ [Function Brief](#)

Function to Fetch a Stream Line by Line

✓ [Function Syntax](#)

```
Output = StreamFetchNext(Stream, ReadSize)
```

✓ [Function Arguments](#)

1. Stream Name
2. Read Size

✓ [Example](#)

```
Text      @varText       ← Variable Declarations
Int       varSize
Stream    @varStream     ← Stream Declaration
```

```
varText = StreamFetchNext(varStream, varSize)
                                     ← Fetching from Stream
```

### **StreamFetchOver**

✓ [Function Brief](#)

Function to Terminate a Stream Fetch.

✓ [Function Syntax](#)

```
StreamFetchOver(Stream)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Argument](#)

1. Stream Name

## **TagParserInit**

### ✓ [Function Brief](#)

Function to initialize a stream for parsing the tags in the stream.

### ✓ [Function Syntax](#)

Output = TagParserInit(stream)

### ✓ [Function Argument](#)

1. Stream Name

### ✓ [Example](#)

```
Stream      @varStream          ← Stream Declaration
Bool        varParse            ← Variable Declaration
varParse = TagParserInit(varStream) ← Initiating Tag Parser
```

varParse will return 1 if tagparsrinit is true.

## **TagParserLocateElement**

### ✓ [Function Brief](#)

Function to locate an element for Parsing

### ✓ [Function Syntax](#)

Output = TagParserLocateElement(stream, "element1.element2.element3")

### ✓ [Function Arguments](#)

1. Stream Name
2. Parse Elements

### ✓ [Example](#)

```
Stream      @varStream          ← Stream Declaration
Bool        varParse            ← Variable Declaration
varParse = TagParserLocateElement(varStream, "root.body.book")
                                                ← Parsing Tag Elements
```

varParse will return 1 if tagparsr locate element is true.

## Vaakya Keywords and Functions Reference Guide

### **TagParserGetElementContent**

✓ [Function Brief](#)

Function to extract the content from an element located.

✓ [Function Syntax](#)

Output = TagParserGetElementContent(stream, sizecontent)

✓ [Function Arguments](#)

1. Stream Name
2. Contents

✓ [Example](#)

```
Stream    @varStream    ← Stream Declaration
Int       varSize      ← Variable Declarations
Text     @varContent
varContent = TagParserElementContent(varStream,varSize)
                                     ← Extracting Contents
```

varContent will return the content of the element located.

### **TagParserGetElementAttribute**

✓ [Function Brief](#)

Function to extract attribute for an element.

✓ [Function Syntax](#)

Output = TagParserGetElementAttribute(stream, attributeName)

✓ [Function Arguments](#)

1. Stream Name
2. Attribute Name

✓ [Example](#)

```
Stream    @varStream    ← Stream Declaration
Int       varSize      ← Variable Declaration
Dict     @varAttribute
```

```
varAttribute = TagParserGetElementAttributes(varStream, "uid")
```

varAttribute will return the value of the uid.

### **TagParserGetElementAttributeList**

✓ [Function Brief](#)

This will get all the attributes of an element located.

✓ [Function Syntax](#)

Output = TagParserGetElementAttributeList(stream)

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Argument](#)

1. Stream Name

### ✓ [Example](#)

```
Stream    @varStream          ← Stream Declaration
Int       varSize             ← Variable Declaration
Dict     @varAttributeList
```

```
varAttributeList = TagParserGetElementAttributes(varStream)
                  ← Getting Element Attributes
varAttributeList will contain all the attributes.
```

## **TagParserClose**

### ✓ [Function Brief](#)

This will close the parser.

### ✓ [Function Syntax](#)

```
TagParserClose(Stream)
```

### ✓ [Function Argument](#)

1. Stream Name

### ✓ [Example](#)

Please create an xml file **library.xml** with the following content.

```
<library libname="newlibrary">
  <book name="book1" author="author1"/>
  <book name="book2" author="author2"/>
  <book name="book3" author="author3"/>
  <book name="book4" author="author4"/>
  <book name="book5" author="author5"/>
</library>
```

The proc to parse the xml file using tagprser keywords is given below.

```
Int    varRet  , varCount , idx
Text   @varlibName, @varBook, @varAuthor, @Namekey,@AuthorKey
```

```
Stream @vsSource          ← Variable Declaration
Dict   @vdAtrbLs, @vdAtrb
vsSource = StreamReadOpenFile("\library.xml", STD_STREAM)
          ← Opening a New Stream
```

```
If(vsSource)
{
  varRet    = 0
  varCount  = 0
  varRet = TagParserInit(vsSource) ← Initialize tag parsing
  varRet = 0
  varRet = TagParserLocateElement(vsSource, "library")
          ← Locating Element "Library"
```

```
If(varRet)
{
  vdAtrb = TagParserGetElementAttribute(vsSource, " libname ")
```

## Vaakya Keywords and Functions Reference Guide

```

    ← Finding attributes libname of the library element.

    If(vdAtrb)
    {
        varlibName = DictFind(vdAtrb, "libname")
        ←Find the libname value from the dict returned
    }
}
varRet = 0
varRet = TagParserLocateElement(vsSource, "library.book")
    ←Locating Element Book
If(varRet)
{
    vdAtrbLs = TagParserGetElementAttributeList(vsSource)
    ← Get Attribute List

    varCount = DictCount(vdAtrbLs)
    varCount /= 2 ← find the number of records available in the dict.Since it
                  contain 2 attributes,name and author. divide the dict
                  count by 2

    idx = 1
    While(idx <= varCount) ← Loop to get individual values
    {
        varBook      =(` `) ← Initializing the variables
        varAuthor    =(` `)

        Namekey      = Concat(idx(06), "name")
        ← Form the key to get the value from dict

        varBook      = DictFind(vdAtrbLs, Namekey)
        ConsolePrint(varBook) ← Print the value

        Authorkey    = Concat(idx(06), "author")
        varAuthor     = DictFind(vdAtrbLs, Authorkey)
        ConsolePrint(varAuthor) ← Print the value
        idx += 1
    }
}
TagParserClose(vsSource)
```

## Vaakya Keywords and Functions Reference Guide

### Utilities

#### Zip

✓ [Function Brief](#)

Function to Zip Text Buffer

✓ [Function Syntax](#)

Output = Zip(src, srcLen, sizZipped)

✓ [Function Arguments](#)

1. Source Text
2. Source Length
3. Zipped Source Size

Function Returns the Zipped buffer.

**Note:**

- Maximum file size to be zipped is 1MB.
- Only text files are allowed to zip in this version.

#### UnZip

✓ [Function Brief](#)

Function to Unzip a text buffer.

✓ [Function Syntax](#)

output = UnZip(src, srcLen, sizZipped)

✓ [Function Arguments](#)

1. Source Text
2. Source Length
3. Zipped Source Size

Output will contain the UnZipped buffer .

## Vaakya Keywords and Functions Reference Guide

### **Encrypt**

✓ [Function Brief](#)

Function to Encrypt a text based on a key.

✓ [Function Syntax](#)

```
Output = Encrypt(Source, Key, SourceSize)
```

✓ [Function Arguments](#)

1. Source Text
2. Encrypt Key
3. Source Size

Function will return the Encrypted value.

**Note:**

- Key Length should be a minimum of 16 or multiples of 16.

### **Decrypt**

✓ [Function Brief](#)

Function to Decrypt a text based on a key.

✓ [Function Syntax](#)

```
Output = Decrypt(Source, Key, SourceSize)
```

✓ [Function Arguments](#)

1. Source Text
2. Encrypt Key
3. Source Size

Output will contain Decrypted Text.

**Note:**

- Key Length should be a minimum of 16 or multiples of 16.

## Vaakya Keywords and Functions Reference Guide

### **Encypher**

Encypher is a more secured encryption.

✓ [Function Brief](#)

Function to encrypt a text based on a key.

✓ [Function Syntax](#)

```
Output = Encypher(Source, Key, SourceSize)
```

✓ [Function Arguments](#)

1. Source Text
2. Encrypt Key
3. Source Size

Output will contain the Encrypted value.

**Note:**

- Key Length should be a minimum of 16 or multiples of 16.

### **Decypher**

✓ [Function Brief](#)

Function to Decypher a text based on a key.

✓ [Function Syntax](#)

```
Output = Decypher(Source, Key, SourceSize)
```

✓ [Function Arguments](#)

1. Source Text
2. Key
3. Source Size

Output will contain the Decrypted value.

**Note:**

- Key Length should be a minimum of 16 or multiples of 16.

### **EncodeB64**

✓ [Function Brief](#)

Function to Encode Source Text to Base64.

✓ [Function Syntax](#)

```
Output = EncodeB64(Source, SourceSize)
```

✓ [Function Arguments](#)

1. Source Text
2. Size to be Encoded.

Output will contain the Encoded value.

## Vaakya Keywords and Functions Reference Guide

### **DecodeB64**

✓ [Function Brief](#)

Function to Decode Source Text.

✓ [Function Syntax](#)

```
Output = DecodeB64(Source, SourceSize)
```

✓ [Function Arguments](#)

1. Source Text
2. Decode Length

Output will contain the Decoded value.

### **HttpEscape**

✓ [Function Brief](#)

Function to remove special characters or spaces from the values which are originating from an http request.

✓ [Function Syntax](#)

```
Output = HttpEscape(Source)
```

✓ [Function Arguments](#)

1. Source Text

Output will contain the value without the space or special characters.

### **HttpUnEscape**

✓ [Function Brief](#)

Function to add special characters or spaces into the the values which have been removed using HttpEscape.

✓ [Function Syntax](#)

```
Output = HttpUnEscape(Source)
```

✓ [Function Arguments](#)

1. Source Text

Output will contain the original value.

### **Ascii2Hex**

This will convert an ascii value in to hexa decimal value.

✓ [Function Brief](#)

Function to convert an ascii value in to hexa decimal value.

✓ [Function Syntax](#)

```
Output = Ascii2Hex(Source)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Arguments](#)

1. Source Text

Output will be an allocated text pointer which contains the hexa decimal value.

## **Hex2Ascii**

This will convert an ascii value in to hexa decimal value.

### ✓ [Function Brief](#)

Function to convert an ascii value in to hexa decimal value.

### ✓ [Function Syntax](#)

```
Output = Hex2Ascii(Source)
```

### ✓ [Function Arguments](#)

1. Source Text

output will be an allocated text pointer which contains ascii value.

## **GetCurrentTime**

Returns the current time along with current date

### ✓ [Function Brief](#)

Function returns the current time along with current date

### ✓ [Function Syntax](#)

```
Output = GetCurrentTime()
```

### ✓ [Function Arguments](#)

- o None

### ✓ [Example](#)

```
Text      @varTime  
varTime = GetCurrentTime()
```

varTime returns the current date and time in `yyymmdd hm` format. (eg. `200907291509`).

## **GetCurrentTimeX**

Returns the current time with milliseconds along with current date

### ✓ [Function Brief](#)

Function returns the current time along with current date

### ✓ [Function Syntax](#)

```
Output = GetCurrentTimeX()
```

### ✓ [Function Arguments](#)

- o None

### ✓ [Example](#)

```
Text      @varTime  
varTime = GetCurrentTimeX()
```

## Vaakya Keywords and Functions Reference Guide

varTime returns the current date and time with milli seconds yyyyymmdd hmsms format.(eg. 20090729151123094).

### **GetCurrentTimeFormatted**

✓ [Function Brief](#)

Function returns the current time along with current date in the format specified.

✓ [Function Syntax](#)

```
Output = GetCurrentTimeFormatted("Format")
```

✓ [Function Argument](#)

- o Format

✓ [Example](#)

```
Text      @varTime
varTime = GetCurrentTimeFormatted("D-M-Y,h:m")
```

varTime returns the current date and time (eg. 29-07-2009,15:15).

### **GetCurrentTimeXFormatted**

✓ [Function Brief](#)

Function reeturns the current time with milliseconds along with current date in a specified format.

✓ [Function Syntax](#)

```
Output = GetCurrentTimeXFormatted("Format")
```

✓ [Function Argument](#)

- o Argument 1
  - 1. Format

✓ [Example](#)

```
Text      @varTime
varTime = GetCurrentTimeXFormatted("D-M-Y,h:m:s:ms")
```

varTime returns the current date and time (eg. 29-07-2009,15:15:10:4800).

### **FileExists**

✓ [Function Brief](#)

Function to find if a File Exists

✓ [Function Syntax](#)

```
Output = FileExists("FilePath")
```

✓ [Function Argument](#)

- 1. FilePath

✓ [Example](#)

```
Int      varRet
varRet = FileExists("\vaakya\test.txt")
```

## Vaakya Keywords and Functions Reference Guide

varRet returns 1 if the file is existing in the given path.

### **FileDelete**

✓ [Function Brief](#)

Function to Delete a file.

✓ [Function Syntax](#)

```
Output = FileDelete("FilePath")
```

✓ [Function Argument](#)

1. FilePath

✓ [Example](#)

```
Int      varRet
varRet = FileDelete("\vaakya\test.txt")
```

varRet returns 1 if the file is deleted.

### **DirectoryCreate**

✓ [Function Brief](#)

Function to create a new directory in a given path.

✓ [Function Syntax](#)

```
Output = DirectoryCreate(FilePath)
```

✓ [Function Argument](#)

1. FilePath

✓ [Example](#)

```
Int      varRet
varRet = DirectoryCreate("\vaakya\test")
```

varRet returns 1 if the directory is created.

### **DirectoryInfo**

✓ [Function Brief](#)

Function which returns a set, containing all the folders and files available in a directory.

✓ [Function Syntax](#)

```
Output = DirectoryInfo(Path, Name, Extension)
```

✓ [Function Argument](#)

1. FilePath  
2. File Name  
3. Extension

✓ [Example](#)

```
DirectoryInfo Recset @dirInfo
Text      @varName, @varExt
```

```
dirInfo = DirectoryInfo("/vaakya/Test", varName, varExt)
```

## Vaakya Keywords and Functions Reference Guide

dirInfo will contain all the files and folder available in /vaakya/Test.

Note:

- If a specific file or folder to be retrieved, mention the name. If it is a file mention the extension also.

### **DirectoryDelete**

✓ [Function Brief](#)

Function to delete a directory in a given path.

✓ [Function Syntax](#)

```
Output = DirectoryDelete(FilePath)
```

✓ [Function Argument](#)

1. FilePath

✓ [Example](#)

```
Int      varRet
varRet = DirectoryDelete("\vaakya\test")
varRet returns 1 if the directory is deleted.
```

### **StoreGlobalPointer**

✓ [Function Brief](#)

Function to Store a Global Pointer which can be accessed globally across all Vaakya Procs. Object, Type, Dict, Array, Recset etc can be stored as a Global Pointer.

✓ [Function Syntax](#)

```
Output = StoreGlobalPointer(Pointer, Key)
```

✓ [Function Argument](#)

1. GlobalPointer
2. Key Value

✓ [Example](#)

```
Int      varRet
Dict     @glbDict

glbDict = New(Dict, 1)

varRet = StoreGlobalPointer(glbDict, "vaakyaglobal")

varRet returns 1 if the global pointer is stored.
```

## Vaakya Keywords and Functions Reference Guide

### **FindGlobalPointer**

✓ [Function Brief](#)

Function to Find a Global Pointer based on a Key.

✓ [Function Syntax](#)

```
Output = FindGlobalPointer(Key)
```

✓ [Function Argument](#)

1. Key

✓ [Example](#)

```
Dict      @glbDict
glbDict = FindGlobalPointer("vaakyaglobal")
glbDict is a pointer which contains the values of the "vaakyaglobal".
```

### **RemoveGlobalPointer**

✓ [Function Brief](#)

Function to remove a Global Pointer based on a key.

✓ [Function Syntax](#)

```
RemoveGlobalPointer(Key)
```

✓ [Function Argument](#)

1. Key

✓ [Example](#)

```
RemoveGlobalPointer("vaakyaglobal")
```

### **Ipcrequest**

✓ [Function Brief](#)

Function to call a Vaakya procedure which is running in another Vaakya application.

✓ [Function Syntax](#)

```
Output=Ipcrequest(Product,corpus,app,domain,procName,reqType,returnTypeName)
```

✓ [Function Arguments](#)

1. Vaakya Product Name
2. Corpus under which the Target App is running
3. Application name
4. Application Partiton under which the procedure has been created.
5. Vaakya Type which has values for the proc to be executed.
6. Vaakya Type into which the results will be filled.

## Vaakya Keywords and Functions Reference Guide

### **HttpRequest**

✓ [Function Brief](#)

This will allow the user to call an external Url from a proc.

✓ [Function Syntax](#)

```
Output = HttpRequest(hostName, port, vsReq)
```

✓ [Function Arguments](#)

1. Host name ( IP Address / Domain Name)
2. Service Port Number
3. URL to be called

### **JsonRequestToSet**

✓ [Function Brief](#)

This will be used when the Json Request is in the form of a set.

✓ [Function Syntax](#)

```
Output = JsonRequestToSet(source)
```

✓ [Function Argument](#)

1. Json Source

✓ [Example](#)

```
Set @varSet  
varSet = JsonRequestToSet(varStream)  
varSet will contain set values.
```

### **JsonRequestToDict**

✓ [Function Brief](#)

This will be used when the Json Request is in the form of a Dict.

✓ [Function Syntax](#)

```
Output = JsonRequestToDict(source)
```

✓ [Function Argument](#)

1. Json Source

✓ [Example](#)

```
Dict @varDict  
varDict = JsonRequestToDict(varStream)  
varDict will contain Dict values.
```

## Vaakya Keywords and Functions Reference Guide

### **JsonRequestToObject**

✓ [Function Brief](#)

This will be used when the Json Request is in the form of an object record.

✓ [Function Syntax](#)

Output = JsonRequestToObject(source)

✓ [Function Argument](#)

1. Json Source

✓ [Example](#)

```
ItemGroup @varGroup
varGroup = JsonRequestToObject(varStream)
varGroup will contain a record.
```

### **JsonRequestToObjectSet**

✓ [Function Brief](#)

This will be used when the Json Request is in the form of a group of records.

✓ [Function Syntax](#)

Output = JsonRequestToObjectSet(source)

✓ [Function Arguments](#)

1. Json Source

✓ [Example](#)

```
ItemGroup Recset @setGroup
setGroup = JsonRequestToObjectSet(varStream)
setGroup will contain a group of records.
```

### **SetToJsonResponse**

✓ [Function Brief](#)

This will take Set values and the response will be of Json format.

✓ [Function Syntax](#)

Output = SetToJsonResponse(setVar, streamVar)

✓ [Function Arguments](#)

1. Set Variable
2. Stream Variable

✓ [Example](#)

```
Set @varSet
Stream @varStream

varStream = WebResponseInit()
SetToJsonResponse(varSet, varStream)
WebResponseSend(varStream)
```

varStream will contain all the values of the set and the response stream will be in json format.

## Vaakya Keywords and Functions Reference Guide

### **DictToJsonResponse**

✓ [Function Brief](#)

This will take Dict values and the response will be of a Json format.

✓ [Function Syntax](#)

Output = DictToJsonResponse(DictVar, streamVar)

✓ [Function Arguments](#)

1. Dictionary Variable
2. Stream Variable

✓ [Example](#)

```
Dict @varDict
Stream      @varStream

varStream = WebResponseInit()
DictToJsonResponse(varDict, varStream)
WebResponseSend(varStream)
```

varStream will contain all the values of the Dict and the response stream will be in json format.

### **ObjectToJsonResponse**

✓ [Function Brief](#)

This will take Object values and the response will be of a Json format.

✓ [Function Syntax](#)

Output = ObjectToJsonResponse(objVar, streamVar)

✓ [Function Arguments](#)

1. Object Variable
2. Stream Variable

✓ [Example](#)

```
ItemGroup @varGroup
varGroup = New(ItemGroup,1)
Stream      @varStream

varStream = WebResponseInit()
ObjectToJsonResponse(varGroup, varStream)
WebResponseSend(varStream)
```

varStream will contain all the values of the ItemGroup Object and the response stream will be in json format.

## Vaakya Keywords and Functions Reference Guide

### **RecsetToJsonResponse**

✓ [Function Brief](#)

This will take Recset values and the response will be of a Json format.

✓ [Function Syntax](#)

```
Output = RecsetToJsonResponse(Recset, streamVar)
```

✓ [Function Arguments](#)

1. Record Set
2. Stream Variable

✓ [Example](#)

```
ItemGroup Recset @setGroup
setGroup = New(Recset, 1)
Stream      @varStream

varStream = WebResponseInit()
RecsetToJsonResponse(setGroup, varStream)
WebResponseSend(varStream)
```

varStream will contain all the values of the ItemGroup Recset and the response stream will be in json format.

## Math Functions

### **Abs**

✓ [Function Brief](#)

Returns absolute value of a given number.

✓ [Function Syntax](#)

```
Output = Abs(signedval)
```

✓ [Example](#)

```
Int      varAbs
varAbs = Abs(-5)
```

varAbs return 5.

### **Floor**

✓ [Function Brief](#)

Returns the lowest value of a given value.

✓ [Function Syntax](#)

```
output = Floor(varNumber)
```

✓ [Example](#)

```
Double varNo
varNo = Floor(10.45)
varNo will contain 10
```

## Vaakya Keywords and Functions Reference Guide

### **Ceil**

✓ [Function Brief](#)

Returns the next highest value of a given value.

✓ [Function Syntax](#)

```
output = Ceil(varNumber)
```

✓ [Example](#)

```
Double varNo  
varNo = Ceil(10.45)  
varNo will contain 11
```

### **Sign**

✓ [Function Brief](#)

Checks the given number is positive or negative. If positive Returns 1.If negative Return -1

✓ [Function Syntax](#)

```
output = Sign(varNumber)
```

✓ [Example](#)

```
Double varNo  
varNo = Sign(-5)  
  
varNo will contain -1.
```

### **Sqrt**

✓ [Function Brief](#)

Returns Square Root of a given number.

✓ [Function Syntax](#)

```
output = Sqrt(varNumber)
```

✓ [Example](#)

```
Double varNo  
varNo = Sqrt(16)  
  
varNo will contain 4.
```

### **Exp**

✓ [Function Brief](#)

Returns Exponent of a given number.

✓ [Function Syntax](#)

```
output = Exp(varNumber)
```

✓ [Example](#)

```
Double varNo  
varNo = Exp(0)
```

## Vaakya Keywords and Functions Reference Guide

varNo will contain 1.

### **Log**

✓ [Function Brief](#)

Returns Natural logarithm of a given number.

✓ [Function Syntax](#)

```
output = Log(varNumber)
```

✓ [Example](#)

```
Double varNo  
varNo = Log(10)
```

varNo will contain 2.30.

### **Log10**

✓ [Function Brief](#)

Returns logarithm of a given number to Base10.

✓ [Function Syntax](#)

```
output = Log10(varNumber)
```

✓ [Example](#)

```
Double varNo  
varNo = Log10(10)
```

varNo will contain 1.

### **Sin**

✓ [Function Brief](#)

Returns Sin of a given number.

✓ [Function Syntax](#)

```
output = Sin(varNumber)
```

✓ [Example](#)

```
Double varNo  
varNo = Sin(90)
```

varNo will contain 0.89(in Radians).

#### **Note:**

- The input value of all trigonometric functions should be in Radians.

### **Cos**

✓ [Function Brief](#)

Returns Cosine value of a given number.

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Syntax](#)

```
output = Cos(varNumber)
```

### ✓ [Example](#)

```
Double varNo  
varNo = Cos(0)  
varNo will contain 1.
```

## **Tan**

### ✓ [Function Brief](#)

Returns Tan of a given number.

### ✓ [Function Syntax](#)

```
output = Tan(varNumber)
```

### ✓ [Example](#)

```
Double varNo  
varNo = Tan(1)  
  
varNo will contain 1.56(in Radians).
```

## **Asin**

### ✓ [Function Brief](#)

Returns Inverse Sin of a given number.

### ✓ [Function Syntax](#)

```
output = Asin(varNumber)
```

### ✓ [Example](#)

```
Double varNo, varNumber  
varNo = Asin(0.87)  
  
varNo will contain 1.06(in Radians).
```

## **Acos**

### ✓ [Function Brief](#)

Returns Inverse Sin of a given number.

### ✓ [Function Syntax](#)

```
output = Asin(varNumber)
```

### ✓ [Example](#)

```
Double varNo, varNumber  
varNo = Acos(0)  
  
varNo will contain 1 (in Radians).
```

## **Atan**

### ✓ [Function Brief](#)

Returns Inverse Tan of a given number.

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Syntax](#)

```
output = Atan(varNumber)
```

### ✓ [Example](#)

```
Double varNo  
varNo = Atan(0.87)  
varNo will contain .72(in Radians).
```

## **Round**

### ✓ [Function Brief](#)

Returns rounded value of a given number for a specified position.

### ✓ [Function Syntax](#)

```
output = Round(varNumber, varDec)
```

### ✓ [Example](#)

```
Int varNo  
varNo = Round(13.54, 1)  
varNo will contain 13.5
```

## **Trunc**

### ✓ [Function Brief](#)

Returns truncated value of a given number.

### ✓ [Function Syntax](#)

```
output = Trunc(varNumber, noofpostion)
```

### ✓ [Example](#)

```
Int varNo  
varNo = Trunc(13.34, 1)  
varNo will contain 13.3.
```

## **Rand**

### ✓ [Function Brief](#)

Returns a random number between range of values.

### ✓ [Function Syntax](#)

```
output = Rand(Low, High)
```

### ✓ [Example](#)

```
Int varNo  
varNo = Rand(1,100)  
  
varNo will return a random number between 1 and 100.
```

## **Min**

### ✓ [Function Brief](#)

Returns minimum of two values.

### ✓ [Function Syntax](#)

```
output = Min(varNumber1, varNumber2)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
Int varNo
varNo = Min(3,2)
varNo will contain 2.
```

## **Max**

### ✓ [Function Brief](#)

Returns maximum of two values.

### ✓ [Function Syntax](#)

```
output = Max(varNumber1, varNumber2)
```

### ✓ [Example](#)

```
Int varNo
varNo = Max(3, 2)
varNo will contain 3.
```

## **Mod**

### ✓ [Function Brief](#)

Returns Modulus of a given number.

### ✓ [Function Syntax](#)

```
output = Mod(varNumber, varDivisor)
```

### ✓ [Example](#)

```
Int varNo
varNo = Mod(10, 3)

varNo will contain 1.
```

## **Pow**

### ✓ [Function Brief](#)

Returns power of a given number.

### ✓ [Function Syntax](#)

```
output = Pow(varNumber, varPow)
```

### ✓ [Example](#)

```
Int varNo
varNo = Pow(2,3)
varNo will contain 8.
```

## Misc Functions

### **DaysBetween**

#### ✓ [Function Brief](#)

Returns the no of days elapsed between two dates.

#### ✓ [Function Syntax](#)

```
ReturnValue = DaysBetween(StartDate, EndDate)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
TextDate StartDate, EndDate
Int      NoOfDays
StartDate = "21052009"
EndDate   = "11052009"
```

```
NoOfDays = DaysBetween(StartDate, EndDate)
```

NoOfDays will contain 10

## **FormDate**

### ✓ [Function Brief](#)

Returns a date after adding or subtracting the number of days .

### ✓ [Function Syntax](#)

```
Output = FormDate(StartDate, NoOfDays)
```

### ✓ [Example](#)

```
TextDate      StartDate
Int           NoOfDays
```

```
StartDate      = "11052005"
NoOfDays       = 10
```

```
EndDate = FormDate(StartDate, NoOfDays)
EndDate will contain 21052005
```

## **DaysInMonth**

### ✓ [Function Brief](#)

Returns number of days in a month.

### ✓ [Function Syntax](#)

```
Output = DaysInMonth(Year, Month)
```

### ✓ [Example](#)

```
Int      varDay
```

```
varDay = DaysInMonth(2009, 02)
varDay will contain 28.
```

## **Concat**

### ✓ [Function Brief](#)

Concat values to a string.

### ✓ [Function Syntax](#)

```
Output = Concat(src1, src2)
```

### ✓ [Example](#)

```
Text      @varConcat
Int       varVal
varVal = 1
```

## Vaakya Keywords and Functions Reference Guide

```
varConcat = Concat("This is",varVal)
varConcat will contain "This is 1"
```

### **ValueInWords**

✓ [Function Brief](#)

Converts a numeric value in to a words.

✓ [Function Syntax](#)

```
Output = ValueInWords(num)
```

✓ [Example](#)

```
Text      @varWords
Int       varVal
varVal = 150
```

```
varWords = ValueInWords(10)
```

varWords will contain "One Hundred and Fifty"

### **EnumValue**

✓ [Function Brief](#)

Returns the value part of an enumerator.

✓ [Function Syntax](#)

```
Output = EnumValue(EnumSet, Label)
```

✓ [Example](#)

```
UOM = {1 = Nos, 2 = Kgs, 3 = Boxes}
Int   varVal
```

```
varVal = EnumValue("UOM", "Kgs")
```

varVal will contain 2.

### **EnumLabel**

✓ [Function Brief](#)

Returns the text part of an enumerator.

✓ [Function Syntax](#)

```
Output = EnumLabel(EnumSet, Value)
```

✓ [Example](#)

```
UOM = {1 = Nos, 2 = Kgs, 3 = Boxes}
Text @varText
```

```
varText = EnumLabel("UOM", 2)
```

varText will contain "Kgs"

## Vaakya Keywords and Functions Reference Guide

### DayofWeek

✓ [Function Brief](#)

Returns the day of the week.

✓ [Function Syntax](#)

```
Output = DayofWeek(Year, Month, Day)
```

✓ [Example](#)

```
TextDate      StartDate  
Int           varDay
```

```
StartDate      = "01012008"  
varDay = DayOfWeek(2008, 01, 01)
```

varDay will contain 2.

**Note:** Day number starts with 0.

### DayName

✓ [Function Brief](#)

Returns the name of the day in a week.

✓ [Function Syntax](#)

```
Output = DayName(Year, Month, Day)
```

✓ [Example](#)

```
TextDate      StartDate  
Text          @varDay  
StartDate      = "01012008"
```

```
varDay = DayName(2008, 01, 01)  
varDay will contain Tuesday.
```

### TextRemoveLineChar

✓ [Function Brief](#)

Removes the line characters from a text.

✓ [Function Syntax](#)

```
Output = TextRemoveLineChar(src)
```

### ConsolePrint

✓ [Function Brief](#)

Prints value in the Console. The Values will be printed in the TraceLog. The User can download the tracelog though the IDE.

✓ [Function Syntax](#)

```
ConsolePrint(Src)
```

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
Int    varVal
varVal = 50
ConsolePrint("Test Print")
ConsolePrint(varVal)
```

## **BitSet**

### ✓ [Function Brief](#)

This will set the value of bit in a particular position from 0 to 1.

### ✓ [Function Syntax](#)

```
Output = BitSet(val,bitno)
bitno is the position of the bit to be converted.
```

Output will be 1 if the bit is set.

## **BitClear**

### ✓ [Function Brief](#)

This will clear the value of bit in a particular position from 1 to 0.

### ✓ [Function Syntax](#)

```
Output = BitClear(val,bitno)
bitno is the position of the bit to be cleared.
```

Output will be 1 if the bit is cleared.

## **BitTest**

### ✓ [Function Brief](#)

This will check the value of a bit in a particular position is 0 or 1.

### ✓ [Function Syntax](#)

```
Output = BitTest(val,bitno)
bitno is the position of the bit to be tested.
```

Output will be 1 or 0.

## **BitFlip**

### ✓ [Function Brief](#)

This will flip the value of bit in a particular position from 1 to 0 or 0 to 1.

### ✓ [Function Syntax](#)

```
Output = BitFlip(val,bitno)
bitno is the position of the bit to be flipped.
```

Output will be 1 if the bit is flipped.

## **SerialPortRead**

### ✓ [Function Brief](#)

This will read the data coming from a serial port.

## Vaakya Keywords and Functions Reference Guide

### ✓ [Function Syntax](#)

```
Output = SerialPortRead(compName, buf, rdMax)
```

compname is the name of the port  
buf is the incoming data buffer.  
rdMax is the number of digits read.

## **SerialPortWrite**

### ✓ [Function Brief](#)

This will write the data to a serial port.

### ✓ [Function Syntax](#)

```
Output = SerialPortWrite(compName, buf, wrMax)
```

compname is the name of the port  
buf is the data buffer to be written.  
wrMax is the number of digits to be written. This will be a user input.

## Web Functions

## **ComposerInit**

### ✓ [Function Brief](#)

Initialize the web template for merging data to Vaakya template variables or Vaakya Block variables

### ✓ [Function Syntax](#)

```
Output = ComposerInit(templateName, header, TRUE/FALSE)
```

Templatename is the name of the template to be initialized.

header should be "HTTP/1.1 200 OK\nContent-Type: text/html\n\n"

TRUE/FALSE has to be set according to the requirement. TRUE will initialize the template only once. FALSE will re-initialize the template. This can be applied for sending templates with dynamic values in email.

Output will be 0 or 1.

## **ComposerSubstituteRecord**

### ✓ [Function Brief](#)

Substitute the values of Vaakya template variables in web template with a type or an object.

### ✓ [Function Syntax](#)

```
Output = ComposerSubstituteRecord(templateName, type/objectVariable)
```

Output will be 0 or 1.

## Vaakya Keywords and Functions Reference Guide

### **ComposerSubstituteVariable**

✓ [Function Brief](#)

This will substitute a value with a variable in a template.

✓ [Function Syntax](#)

Output = ComposerSubstituteVariable(templateName, variable)

Output will be 0 or 1.

### **ComposerSubstituteBlock**

✓ [Function Brief](#)

This will substitute a group of records in a template.

✓ [Function Syntax](#)

Output = ComposerSubstituteBlock(templateName, blkName, type/objectVariable)

Output will be 0 or 1.

### **ComposerSave**

✓ [Function Brief](#)

This will save a template, with all the values merged, at runtime.

✓ [Function Syntax](#)

Output = ComposerSave(templateName, footer, fileName)

Footer will be a text pointer.

FileName is the name of the file to be saved.

Output will be 0 or 1.

### **ComposerSend**

✓ [Function Brief](#)

This will send the merged contents to response stream.

✓ [Function Syntax](#)

Output = ComposerSend(templateName, footer)

Footer will be a text pointer.

Output will be 0 or 1.

## Vaakya Keywords and Functions Reference Guide

### **WebResponseInit**

✓ [Function Brief](#)

This will be used to initialize a response stream as a Json response.

✓ [Function Syntax](#)

Output = WebResponseInit()

✓ [Example](#)

```
Stream      @varStream
varStream = WebResponseInit()
```

### **WebResponseSend**

✓ [Function Brief](#)

This will be used to send a response stream as a Json response or to the Browser.

✓ [Function Syntax](#)

Output = WebResponseSend()

✓ [Example](#)

```
Stream      @varStream

varStream    = WebResponseInit()

WebResponseSend(varStream)
```

### **ConvertMimeDataToRecord**

✓ [Function Brief](#)

This will map mime data to a type/object.

✓ [Function Syntax](#)

```
Output = ConvertMimeDataToRecord(vaMpdf, "type")
        ← default Mime data is received in vaMpdf
```

All the values of vaMpdf will be mapped to the type.

## Communication Functions

### **MailBoxInit**

✓ [Function Brief](#)

This will initialize the mailbox for sending or receiving emails.

✓ [Function Syntax](#)

Output = MailBoxInit(MailServerIp, UserName, Password)

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
SysHandle      @ptrSys
VapMail        ptrMail
ptrSys         = MailBoxInit("7.6.5.4", "test", "testpasswd")
```

## MailDespatch

### ✓ [Function Brief](#)

Mail despatch will send a mail from the application if the mail server configuration is correct.

### ✓ [Function Syntax](#)

```
MailDespatch(SysPointer, MailPointer)
```

### ✓ [Example](#)

```
SysHandle @ptrSys

VapMail          ptrMail
ptrMail          = New(VapMail, 1)
ptrSys           = MailBoxInit("7.6.5.4", "test", "testpasswd")
ptrMail.from     = Concat("testuser@mail.com")
ptrMail.to       = Concat("receiver@mail.com")
ptrMail.contentType = Concat("text/plain")
ptrMail.xMailer  = Concat("testMailer")
ptrMail.subject  = Concat("test subject")
ptrMail.content  = Concat("This is a test message")

MailDespatch(ptrSys, ptrMail)
```

**Note:** xMailer will be the email client.

## MailCheck

### ✓ [Function Brief](#)

This will return the number of unread message in the initialized mailbox.

### ✓ [Function Syntax](#)

```
MailCheck(SysPointer)
```

### ✓ [Example](#)

```
SysHandle      @ptrSys
Int            Count
ptrSys         = MailBoxInit("7.6.5.4", "recv", "recvpaswd")
Count         = MailCheck(ptrSys)
```

## Vaakya Keywords and Functions Reference Guide

### MailReceive

✓ [Function Brief](#)

Function to receive Email into a Mail Box

✓ [Function Syntax](#)

MailReceive(SysPointer, persistvariable)

✓ [Example](#)

```
SysHandle      @ptrSys
Int            Count

ptrSys         = MailBoxInit("7.6.5.4", "recv", "recvpasswd")
Count         = MailCheck(ptrSys)

If(Count)
{
    retMail      = MailRecv(ptrSys, mailpersist)
}
```

**Note:** mailpersist will be 0 or 1. If it is 0 the mail will be retained after reading. If it is 1 the mail will be deleted after reading.

### SMSBoxInit

✓ [Function Brief](#)

This will initialize the sms box for sending or receiving sms.

✓ [Function Syntax](#)

Output = SmsBoxInit(Port, PortAttributes, Mode)

✓ [Example](#)

```
SysHandle      @ptrSys
VapSms        @ptrSms
Text          @varPort, @varPortAttr

varPort       = Concat("COM1")
varPortAttr   = Concat("9600, 8, N, 0")

ptrSys        = SmsBoxInit(varPort, varPortAttr, 1)
```

**Note:** Mode can be 0 or 1. Zero is PDU mode and one is sms.

### SmsDespatch

✓ [Function Brief](#)

Function to Send an SMS

✓ [Function Syntax](#)

SmsDespatch(SysPointer, SMSPointer)

## Vaakya Keywords and Functions Reference Guide

### ✓ [Example](#)

```
Int          varRet, varMode
SysHandle    @ptrSys
VapSms       @ptrSms
ptrSms       = New(VapSms, 1)
varRet       = 0
varMode      = 0

Text         @varPort, @varPortAttr, @mobile, @message
varPort      = Concat("COM1")
varPortAttr  = Concat("9600, 8, N, 0")

ptrSys       = SmsBoxInit(varPort, varPortAttr, varMode)

mobile       = Concat("+919880012345")
message      = "This is a test message"

ptrSms.phoneNumber = mobile
ptrSms.message     = message

varRet = SmsDespatch(ptrSys, ptrSms)
```

## **SmsReceive**

### ✓ [Function Brief](#)

Function to Receive an SMS

### ✓ [Function Syntax](#)

SmsReceive(SysPointer, Mod)

### ✓ [Example](#)

```
Int          varMode
SysHandle    @ptrSys
VapSms       ptrRecvSms

varMode      = 1
ptrRecvSms   = SmsRecv(ptrSys, varMode)
```

## Vaakya Keywords and Functions Reference Guide

### Contact Us

#### Corporate Office:

601, First Floor,  
12th Main, HAL 2nd Stage,  
Indiranagar, Bangalore  
PIN - 560008  
India

[feedback@vaakya.com](mailto:feedback@vaakya.com)

### Copyright

"The copyright of any and all material contained herein is owned and reserved by VAAKYA TECHNOLOGIES PRIVATE LIMITED ("**OWNER**") except where otherwise stated. Any reproduction, copying and/or distribution in any form of the material, in whole or in part, are not permitted without prior written consent from the OWNER. Trademarks, logos, images, text or content of third parties ("**THIRD PARTY PROPERTY**") used herein are the property of their respective owners, and have been used without permission except where otherwise indicated. The Owner makes no claim to such THIRD PARTY PROPERTY used herein. All use of THIRD PARTY PROPERTY contained herein is for non-profit purposes only, for the proper guidance of the users, considering the convenience and familiarity of the user in associating the THIRD PARTY PROPERTY with the owners of THIRD PARTY PROPERTY and not for advertisement or establishing any connection with the owners of THIRD PARTY PROPERTY. Other than THIRD PARTY PROPERTY, all trademarks, tradenames, logos, designs and all related product and service names are the sole property of the OWNER, and may not be used in any manner without the prior written consent of the OWNER. The materials contained herein may include inaccuracies or typographical errors and the owner shall not be held responsible for the same. The Owner reserves the sole right to modify, amend, delete, omit, edit or include any material or make periodic changes to the material herein.