Visual Fairy for Applications  
Reference Manual  
Version 1.0  
Technical Preview

TABLE OF CONTENTS

[1. Overview 3](#_Toc141799224)

[2. Basic program structure 3](#_Toc141799225)

[3. Types 3](#_Toc141799226)

[3.1. Booleans 3](#_Toc141799227)

[3.2. Integers 3](#_Toc141799228)

[3.3. Strings 4](#_Toc141799229)

[3.4. Arrays of integers 4](#_Toc141799230)

[4. Variables 4](#_Toc141799231)

[5. Functions 5](#_Toc141799232)

[6. Conditional control flow 5](#_Toc141799233)

[7. Builtin functions 6](#_Toc141799234)

[7.1. Type conversion 6](#_Toc141799235)

[7.2. String operations 6](#_Toc141799236)

[7.3. Debugging 7](#_Toc141799237)

[7.4. Random number generation 7](#_Toc141799238)

[7.5. Document sharing 7](#_Toc141799239)

[8. Security 7](#_Toc141799240)

[9. Execution environment 8](#_Toc141799241)

# Overview

Visual Fairy for Applications (VFA) is a brand-new language for writing MaScroll document macros. Designed and implemented by our multi-kingdom industry-leading team of management wizards and our one part-time CPU spell engineering intern, VFA allows you to automate your spell recipe transformations so that you can infinitely scale your number of active overdue projects and maximize wand-holder profits.

# Basic program structure

A VFA program receives the original document text, transforms it, and produces a new version of the text. The entry point is the Main function, which receives the original text and returns the transformed one.

The following example returns the original text unchanged:

Function Main(text As String) As String

Main = text

End Function

Single-line comments are initiated by the ' character.

# Types

There are three primitive types: booleans, integers, and strings. It is also possible to create arrays of integers.

## Booleans

The Boolean type can take two literal values: True and False.

The following operators are defined on Booleans:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Arity | Output type | Description |
| Not | Unary | Boolean | Boolean negation |
| And | Binary | Boolean | Boolean conjunction |
| Or | Binary | Boolean | Boolean disjunction |

## Integers

The Integer type is a 64-bit signed integer. Integer literals can be written in decimal (no prefix), octal (&O prefix), or hexadecimal (&H prefix). The following operators are defined on integers:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Arity | Output type | Description |
| - | Unary | Integer | Integer negation |
| Not | Unary | Integer | Bitwise NOT |
| + | Binary | Integer | Addition |
| - | Binary | Integer | Subtraction |
| \* | Binary | Integer | Multiplication |
| / | Binary | Integer | Division |
| And | Binary | Integer | Bitwise AND |
| Or | Binary | Integer | Bitwise OR |
| < | Binary | Boolean | Less than |
| <= | Binary | Boolean | Less than or equal |
| > | Binary | Boolean | Greater than |
| >= | Binary | Boolean | Greater than or equal |
| = | Binary | Boolean | Equal to |
| <> | Binary | Boolean | Not equal to |

## Strings

The String type represents a sequence of bytes. String literals are enclosed in double quotes (") and can contain the escape sequences \t, \r, \n, and \\.

The following operators are defined on strings (comparisons are done lexicographically):

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Arity | Output type | Description |
| & | Binary | String | String concatenation |
| < | Binary | Boolean | Less than |
| <= | Binary | Boolean | Less than or equal |
| > | Binary | Boolean | Greater than |
| >= | Binary | Boolean | Greater than or equal |
| = | Binary | Boolean | Equal to |
| <> | Binary | Boolean | Not equal to |

## Arrays of integers

Arrays of integers can be created when declaring variables (see Variables). The subscript operator is (). Indexes are zero-based. For example, arr(2) references the third element in the array arr.

# Variables

Variables can be declared anywhere in the program with the following syntax:

' Declare a variable named <identifier> of type <type>

Dim <identifier> As <type>

' Declare an array of integers of length <length>

Dim <identifier>(<length>) As Integer

**WARNING:** the array declaration takes the length, not the index upper bound. This is a common mistake when migrating macros from non-magical languages.

Variables defined in inner blocks shadow same-name variables defined in outer scopes.

Variables can be assigned using the = operator. Variables are automatically initialized on declaration to False for booleans, 0 for integers, and "" for strings. All elements in integer arrays are initialized to 0.

# Functions

Functions are defined using the following syntax:

' Define a function named <identifier> returning <return-type>

' Parameter N named <param-N> of type <type-N>

Function <identifier>(<param-1> As <type-1>[, <param-2> As <type-2>[, ...]]) As <return-type>

...

End Function

Setting the return value is achieved by setting the function variable:

Function Add(a As Integer, b As Integer) As Integer

Add = a + b

End Function

The syntax for accepting array parameters is as follows:

Function GetFirst(arr() As Integer) As Integer

GetFirst = arr(0)

End Function

Nested functions are not allowed.

Each parameter declaration may be prefixed with ByVal (the default) or ByRef to indicate whether the argument should be passed by value or reference. For example:

Function g(ByRef x As Integer) As Integer

x = 42

g = 0

End Function

Function f() As Integer

Dim result As Integer

g(result)

' result is now 42

f = result

End Function

# Conditional control flow

The following conditional control flow structures are supported:

' <condition> is a boolean expression

If <condition> Then

...

End If

While <condition>

...

End While

# Builtin functions

## Type conversion

CBool(value As Boolean|Integer|String) As Boolean

Converts value to a boolean. Integers are converted to True when non-zero, False when zero. Strings are converted to True when non-empty, False when empty.

CInt(value As Boolean|Integer|String) As Integer

Converts value to an integer. True is converted to 1, False to zero. Strings are interpreted as integer literals.

CStr(value As Boolean|Integer|String|Integer()) As String

Converts value to a string representation.

## String operations

Asc(str As String) As Integer

Returns the integer value of the first byte of str.

Chr(value As Integer) As String

Returns a 1-byte string containing the byte value.

Len(str As String) As Integer

Returns the length of str.

Mid(str As String, start As Integer[, len As Integer]) As String

Returns the substring of str of length len starting at position start (1-based). If len is omitted, the substring spans until the end of str.

String(len As Integer, value As Integer|String) As String

Returns a string of length len obtained by repeating the byte value (if integer) or the first byte of value (if string).

## Debugging

Describe(value) As String

Returns a string representation of the type of value.

## Random number generation

CryptoRand([ByRef str As String[, entropy As String]]) As Integer

When invoked with no parameters, returns a random integer. When invoked with one or two parameters, fills str with random bytes and returns its length. If entropy is specified, its contents are mixed into the random generation as an additional source of entropy. The randomness produced by this function is suitable for cryptographic purposes.

FastRand([ByRef str As String[, entropy As String]]) As Integer

Same semantics as CryptoRand. This random generator is faster, however, the resulting randomness is not suitable for cryptographic purposes and may not meet stringent distribution requirements.

RandSeed(seed As Integer) As Integer

Seeds the generator used by FastRand (no effect on CryptoRand). Returns the seed.

## Document sharing

SignToken(documentId As String) As String

Returns a MaScroll shared link signature for the document with ID documentId. The current user must be the owner of the document.

# Security

This technical preview is experimental and could be vulnerable to malicious spells. We encourage third-party security researchers to report any vulnerability they find. While we cannot share source code, this preview is built with low optimizations, no inlining, and includes PDB debug information to ease reverse engineering. Internal security review is ongoing. Auditing on the parsing code has been completed and we have no knowledge of outstanding vulnerabilities in parsing. Therefore, we advise third-party researchers to investigate other components, as they have been subject to less security scrutiny as of the time of writing.

# Execution environment

This technical preview is pre-packaged in a Windows Server virtual machine. The VFA interpreter does not have strict environment requirements, but we advise to not update the Windows OS during critical working hours to minimize the chance of unforeseen breakage.