MMBasic Structures User Manual

Introduction

Structures (also known as User-Defined Types) allow you to group related variables of different types together under a single name. This is useful for organizing complex data such as coordinates, records, or any collection of related values.

Defining a Structure Type

Use the "TYPE...END TYPE" block to define a new structure type:

```
  Type typename
    member1 As type
    member2 As type
    ...
  End Type
```

Supported Member Types

- "INTEGER" - 64-bit signed integer
- "FLOAT" - 64-bit floating point number
- "STRING" - String up to 255 characters (use "LENGTH n" to specify maximum length)

Examples

Simple structure:

```
  Type Point
    x As INTEGER
    y As INTEGER
  End Type
```

Structure with mixed types:

```
  Type Person
    age As INTEGER
    height As FLOAT
    name As STRING
  End Type
```

Structure with string length specified:

```
  Type Record
    id As INTEGER
    description As STRING LENGTH 100
  End Type
```

Memory Layout and Alignment

Understanding how structures are packed in memory is important when working with binary file I/O or calculating memory usage.

Member Storage Sizes

Type	Storage Size
`INTEGER`	8 bytes
`FLOAT`	8 bytes
`STRING`	length + 1 bytes (1 byte for length prefix + specified/default length)
`STRING LENG...	n + 1 bytes
Nested struc...	Size of the nested structure type

Alignment Rules

Members are placed sequentially in memory with the following alignment rules:
- **Strings**: No alignment requirement. Placed immediately after the previous member.
- **INTEGER, FLOAT, and nested structures**: Aligned to 8-byte boundaries. If the current offset is not divisible by 8, padding bytes are inserted before the member.

Padding Example

When a numeric type follows a string whose total storage is not aligned to 8 bytes, padding is automatically inserted:

```
  Type Example1
    name As STRING LENGTH 10   ' Offset 0, size 11 bytes (10 + 1 length byte)
    value As INTEGER           ' Offset 16 (padded from 11 to align to 8)
  End Type
  ' Total size: 24 bytes (11 + 5 padding + 8)

  Type Example2
    name As STRING LENGTH 15   ' Offset 0, size 16 bytes (15 + 1 length byte)
    value As INTEGER           ' Offset 16 (no padding needed, already aligned)
  End Type
  ' Total size: 24 bytes (16 + 8)

  Type Example3
    a As INTEGER               ' Offset 0, size 8 bytes
    name As STRING LENGTH 5    ' Offset 8, size 6 bytes (5 + 1 length byte)
    b As INTEGER               ' Offset 16 (padded from 14 to align to 8)
  End Type
  ' Total size: 24 bytes (8 + 6 + 2 padding + 8)
```

Optimizing Structure Size

To minimize wasted space from padding, consider:
1. **Grouping numeric members together**: Place all INTEGER and FLOAT members consecutively.
2. **Using string lengths that result in 8-byte aligned totals**: String lengths of 7, 15, 23, 31, etc. (where length + 1 is divisible by 8) avoid padding when followed by numeric types.

Note on structure end padding: Padding is always added to the end of a structure to ensure the total size is aligned to 8 bytes. This is required so that arrays of structures maintain proper memory alignment for all elements. Without end padding, the second element of an array would start at a misaligned address, potentially causing memory access errors.

```
  ' Less efficient (has internal padding):
  Type Inefficient
    flag As INTEGER            ' 8 bytes
    name As STRING LENGTH 10   ' 11 bytes
    count As INTEGER           ' 8 bytes (but needs 5 bytes padding before it)
  End Type
  ' Total: 32 bytes

  ' More efficient (no internal padding, but end padding still applies):
  Type Efficient
    flag As INTEGER            ' 8 bytes
    count As INTEGER           ' 8 bytes
    name As STRING LENGTH 10   ' 11 bytes + 5 bytes end padding
  End Type
  ' Total: 32 bytes (padded to 8-byte boundary)
```

Use "STRUCT(SIZEOF, "typename")" to verify the actual size of your structures.

Declaring Structure Variables

Use "DIM" to declare variables of a structure type:

```
  Dim variablename As typename
```

Examples

Simple structure variable:

```
Type Point
  x As INTEGER
  y As INTEGER
End Type

Dim p As Point
Dim origin As Point
```

Multiple variables:

```
Dim p1 As Point, p2 As Point, p3 As Point
```

Accessing Structure Members

Use the dot (".") notation to access individual members:

```
variablename.membername
```

Examples

Setting values:

```
Dim p As Point
p.x = 100
p.y = 200
```

Reading values:

```
Print p.x, p.y
result = p.x + p.y
```

Using in expressions:

```
distance = Sqr(p.x * p.x + p.y * p.y)
```

Arrays of Structures

You can create arrays where each element is a structure:

```
Dim arrayname(size) As typename
```

Examples

Declaring an array of structures:

```
Dim points(10) As Point
```

Accessing array elements:

```
points(0).x = 10
points(0).y = 20
points(1).x = 30
points(1).y = 40

Print points(0).x, points(0).y
```

Using variable indices:

```
For i = 0 To 10
  points(i).x = i * 10
  points(i).y = i * 20
Next i
```

Multi-dimensional arrays:

```
Dim grid(10, 10) As Point
grid(5, 5).x = 100
grid(5, 5).y = 200
```

Initializing Structures

Structures can be initialized when declared using parentheses with comma-separated values:

```
Dim variablename As typename = (value1, value2, ...)
```

Values must be provided in the order the members are defined in the TYPE block.

Examples

Simple structure initialization:

```
Type Point
  x As INTEGER
  y As INTEGER
End Type

Dim p As Point = (100, 200)
' p.x = 100, p.y = 200
```

Structure with string:

```
Type Person
  age As INTEGER
  height As FLOAT
  name As STRING
End Type

Dim person1 As Person = (25, 1.75, "Alice")
```

Array of structures initialization:

```
Dim points(2) As Point = (10, 20, 30, 40, 50, 60)
' points(0).x = 10, points(0).y = 20
' points(1).x = 30, points(1).y = 40
' points(2).x = 50, points(2).y = 60
```

Values are assigned sequentially: all members of element 0, then all members of element 1, etc.

Copying Structures

Structures can be copied using direct assignment or the "STRUCT COPY" command.

Direct Assignment

The simplest way to copy a structure is using direct assignment:

```
destination = source
```

This works for:

- Single structure variables
- Individual array elements

Example - Single structures:

```
Dim src As Point, dst As Point
src.x = 100
src.y = 200

dst = src
' dst.x = 100, dst.y = 200
```

Example - Array elements:

```
Dim points(10) As Point
points(0).x = 50 : points(0).y = 60
points(5) = points(0)    ' Copy element 0 to element 5
' points(5).x = 50, points(5).y = 60
```

Example - Cross-array copy:

```
Dim src(5) As Person
Dim dst(5) As Person
' ... populate src ...

dst(2) = src(3)    ' Copy element 3 from src to element 2 of dst
```

Important: Both source and destination must be the same structure type. Attempting to assign structures of different types will cause an error.

STRUCT COPY Command

The "STRUCT COPY" command provides the same functionality with explicit syntax:

```
Struct Copy source To destination
```

Both variables must be of the same structure type.

Example:

```
Dim src As Point, dst As Point
src.x = 100
src.y = 200

Struct Copy src To dst
' dst.x = 100, dst.y = 200
```

Copying Entire Arrays

You can copy entire structure arrays using empty parentheses:

```
Struct Copy sourceArray() To destinationArray()
```

Requirements:

- Both arrays must be the same structure type
- The destination array must be at least as large as the source array
- Both must use the "()" syntax, or both must be single elements
- Only the source elements are copied (extra destination elements are preserved)

Example:

```
Dim src(2) As Point
src(0).x = 10 : src(0).y = 11
src(1).x = 20 : src(1).y = 21
src(2).x = 30 : src(2).y = 31

Dim dst(4) As Point  ' Larger destination is OK
Struct Copy src() To dst()
' dst(0), dst(1), dst(2) now contain copies from src
' dst(3), dst(4) are unchanged
```

Example - Same size arrays:

```
Dim original(10) As Person
' ... populate array ...

Dim backup(10) As Person
Struct Copy original() To backup()
```

Sorting Structure Arrays

Use the "STRUCT SORT" command to sort an array of structures in-place based on any member field:

```
Struct Sort array(), membername [, flags]
```

Parameters

- "array()" - The structure array to sort (must include empty parentheses)
- "membername" - The name of the member field to sort by
- "flags" - Optional flags to modify sort behavior (can be combined by adding):

Value	Description
:-----:	-------------
0	Default: ascending sort, case-sensitive
1	Reverse sort (descending order)
2	Case-insensitive sort (strings only)
4	Empty strings sort to end of array (strings only)

Flags can be combined by adding values (e.g., 3 = descending + case-insensitive).

Examples

Sort by integer field (ascending):

```
Type Person
  age As INTEGER
  name As STRING
End Type

Dim people(3) As Person
people(0).age = 35 : people(0).name = "Charlie"
people(1).age = 25 : people(1).name = "Alice"
people(2).age = 45 : people(2).name = "David"
people(3).age = 30 : people(3).name = "Bob"

Struct Sort people(), age
' Result: Alice(25), Bob(30), Charlie(35), David(45)
```

Sort by string field:

```
Struct Sort people(), name
' Result: Alice, Bob, Charlie, David
```

Reverse sort (descending):

```
Struct Sort people(), age, 1
' Result: David(45), Charlie(35), Bob(30), Alice(25)
```

Case-insensitive string sort:

```
Struct Sort people(), name, 2
```

Combine flags (reverse + case-insensitive):

```
Struct Sort people(), name, 3
' 3 = 1 + 2 (reverse + case insensitive)
```

Empty strings at end:

```
Struct Sort people(), name, 4
' Non-empty strings sorted first, empty strings at end
```

Notes

- The entire structure is moved during sorting, not just the sort key
- All other member values are preserved with their corresponding records
- The sort is performed in-place (no additional array is created)
- Array members within structures cannot be used as sort keys

- Supports INTEGER, FLOAT, and STRING member types

Clearing Structures

Use the "STRUCT CLEAR" command to reset all members of a structure to their default values (0 for numbers, empty string for strings):

```
Struct Clear variable
Struct Clear array()
```

Examples

Clear a single structure:

```
Dim p As Point
p.x = 100
p.y = 200

Struct Clear p
' p.x = 0, p.y = 0
```

Clear an entire array of structures:

```
Dim people(10) As Person
' ... populate array ...

Struct Clear people()
' All elements reset to defaults
```

Swapping Structures

Use the "STRUCT SWAP" command to exchange the contents of two structure variables:

```
Struct Swap var1, var2
```

Both variables must be of the same structure type. This is useful when implementing sorting algorithms or reordering records.

Examples

```
Dim a As Point, b As Point
a.x = 10 : a.y = 20
b.x = 30 : b.y = 40

Struct Swap a, b
' Now: a.x = 30, a.y = 40, b.x = 10, b.y = 20
```

Swapping array elements:

```
Dim people(5) As Person
' ... populate array ...

Struct Swap people(2), people(4)
' Elements 2 and 4 are exchanged
```

Printing Structures

Use the "STRUCT PRINT" command to display all members of a structure for debugging:

```
Struct Print variable
Struct Print array()
Struct Print array(index)
```

Forms

- "Struct Print variable" - Print a single structure variable
- "Struct Print array()" - Print all elements of a structure array
- "Struct Print array(n)" - Print a specific element of a structure array

Examples

Print a single structure:

```
Dim p As Person
p.name = "Alice"
p.age = 25
p.height = 1.65

Struct Print p
```

Output:

```
Person:
  .name = "Alice"
  .age = 25
  .height = 1.65
```

Print an array element:

```
Dim people(10) As Person
' ... populate array ...

Struct Print people(0)
```

Print entire array:

```
Struct Print people()
```

Output:

```
Person array (11 elements):
[0]:
  .name = "Alice"
  .age = 25
  .height = 1.65
[1]:
  .name = "Bob"
  .age = 30
  .height = 1.80
...
```

Notes

- Array members are printed as comma-separated values
- Strings are displayed with surrounding quotes
- Useful for debugging and inspecting structure contents

Searching Structure Arrays

Use the "STRUCT(FIND, ...)" function to search a structure array for an element with a matching member value:

```
index = Struct(FIND, array(), membername$, value [, start])
```

Parameters

- "FIND" - The subfunction name
- "array()" - The structure array to search (must include empty parentheses)
- "membername$" - The name of the member field to search (as a string)
- "value" - The value to search for (must match the member's type)
- "start" - Optional. The index to start searching from (default: first element)

Return Value

- Returns the index of the first matching element (starting from "start")
- Returns -1 if no match is found

Examples

Find by integer:

```
Type Person
  age As INTEGER
  name As STRING
End Type

Dim people(10) As Person
' ... populate array ...

idx = Struct(FIND, people(), "age", 35)
If idx >= 0 Then
  Print "Found at index"; idx; ": "; people(idx).name
Else
  Print "Not found"
EndIf
```

Find by string:

```
idx = Struct(FIND, people(), "name", "Alice")
If idx >= 0 Then
  Print "Alice is at index"; idx
EndIf
```

Find by float:

```
idx = Struct(FIND, people(), "height", 1.75)
```

Iterate through all matches using start parameter:

```
' Find all people aged 30
idx = Struct(FIND, people(), "age", 30)
Do While idx >= 0
  Print "Found at index"; idx; ": "; people(idx).name
  idx = Struct(FIND, people(), "age", 30, idx + 1)
Loop
```

Notes

- Search is performed linearly from the start position
- Only the first match (from the start position) is returned
- Use the start parameter to iterate through multiple matches
- For strings, comparison is case-sensitive and exact
- Array members within structures cannot be searched
- The member name is passed as a string (can be a variable)

Getting Array Bounds

Use the "STRUCT(SIZE, ...)" function to get the upper bound of a structure array dimension:

```
upperBound = Struct(SIZE, array() [, dimension])
```

Parameters

- "SIZE" - The subfunction name
- "array()" - The structure array (must include empty parentheses)
- "dimension" - Optional. Which dimension to query (1-based, default: 1)

Return Value

- Returns the upper bound of the specified dimension

Examples

Basic usage (1D array):

```
  Dim points(9) As Point
  bound = Struct(SIZE, points())    ' Returns 9
```

Multi-dimensional array:

```
  Dim grid(4, 7) As Point
  dim1 = Struct(SIZE, grid(), 1)    ' Returns 4
  dim2 = Struct(SIZE, grid(), 2)    ' Returns 7
```

Default dimension:

```
  Dim data(10, 20) As Person
  bound = Struct(SIZE, data())      ' Returns 10 (first dimension)
```

Use in loops:

```
  Dim people(n) As Person
  ' ... populate array ...

  For i = 0 To Struct(SIZE, people())
    Print people(i).name
  Next i
```

Notes

- Similar to the standard BOUND() function but for structure arrays
- Dimension numbering is 1-based (1 = first dimension, 2 = second, etc.)
- If dimension is omitted, returns the bound of the first dimension

Getting Structure Size

Use the "STRUCT(SIZEOF, ...)" function to get the size in bytes of a structure type:

```
  byteSize = Struct(SIZEOF, typename$)
```

Parameters

- "SIZEOF" - The subfunction name
- "typename$" - A string containing the structure type name

Return Value

- Returns the total size in bytes of the specified structure type

Examples

Basic usage:

```
  Type Point
    x As INTEGER
    y As INTEGER
  End Type

  size = Struct(SIZEOF, "Point")    ' Returns 16 (2 x 8-byte integers)
  Print "Point size:"; size; "bytes"
```

Using with different types:

```
  Type Person
    age As INTEGER        ' 8 bytes
    height As FLOAT       ' 8 bytes
    name As STRING        ' 256 bytes (default string)
  End Type

  size = Struct(SIZEOF, "Person")   ' Returns 272
```

Dynamic type name:

```
  typename$ = "Point"
  size = Struct(SIZEOF, typename$)
```

Use for memory calculations:

```
  Dim data(99) As Record
  totalBytes = 100 * Struct(SIZEOF, "Record")
  Print "Array uses"; totalBytes; "bytes"
```

Notes

- The type name comparison is case-insensitive
- Returns an error if the structure type is not defined
- Useful for calculating memory requirements or file sizes before STRUCT SAVE

Saving and Loading Structures

Structures can be saved to and loaded from files in binary format. This is useful for persisting data between program runs or exchanging data.

STRUCT SAVE

Writes structure data to an already-open file:

```
  Struct Save #filenumber, variable
  Struct Save #filenumber, array()
  Struct Save #filenumber, array(index)
```

The file must be opened before using "STRUCT SAVE". You manage the file opening and closing.

Syntax options for arrays:

- "array()" - Saves the entire array
- "array(index)" - Saves only the element at the specified index

STRUCT LOAD

Reads structure data from an already-open file:

```
  Struct Load #filenumber, variable
  Struct Load #filenumber, array()
  Struct Load #filenumber, array(index)
```

The file must be opened before using "STRUCT LOAD". The structure variable must already be declared.

Syntax options for arrays:

- "array()" - Loads the entire array
- "array(index)" - Loads only into the element at the specified index

Examples

Save and load a single structure:

```
  Dim p As Point
  p.x = 100
  p.y = 200
```

```
' Save
Open "point.dat" For Output As #1
Struct Save #1, p
Close #1

' Load
Dim p2 As Point
Open "point.dat" For Input As #1
Struct Load #1, p2
Close #1

Print p2.x, p2.y   ' Output: 100    200
```

Save and load an array of structures:

```
Dim people(100) As Person
' ... populate array ...

' Save to file
Open "people.dat" For Output As #1
Struct Save #1, people()
Close #1

' Load from file
Dim loadedPeople(100) As Person
Open "people.dat" For Input As #1
Struct Load #1, loadedPeople()
Close #1
```

Save and load individual array elements:

```
Dim records(99) As Record
' ... populate records ...

' Save specific records to file
Open "selected.dat" For Output As #1
Struct Save #1, records(5)    ' Save element 5
Struct Save #1, records(10)   ' Append element 10
Struct Save #1, records(25)   ' Append element 25
Close #1

' Load records back (to different positions or variables)
Dim temp As Record
Open "selected.dat" For Input As #1
Struct Load #1, temp          ' Load first saved record
Print temp.id
Struct Load #1, records(50)   ' Load second saved record into element 50
Struct Load #1, records(51)   ' Load third saved record into element 51
Close #1
```

Notes

- Data is saved in binary format (not human-readable)
- The structure type must match when loading
- For whole array operations, array dimensions must match when loading
- Files should be opened in appropriate mode for the operation
- Multiple structures can be saved to the same file sequentially
- Error occurs if file is not open or is not a disk file
- **Array variables must use parentheses**: "array()" for whole array, "array(i)" for single element
- Using an array name without parentheses will cause an error

Structures in Subroutines and Functions

Passing Structures as Parameters

Structures are always passed **by reference**, meaning the subroutine or function can modify the original structure:

```
Sub subname(parametername As typename)
  ...
End Sub
```

Example - Read-only access:

```
Sub PrintPoint(pt As Point)
  Print "X:"; pt.x; " Y:"; pt.y
End Sub

Dim p As Point = (100, 200)
PrintPoint p
```

Example - Modifying the structure:

```
Sub DoublePoint(pt As Point)
  pt.x = pt.x * 2
  pt.y = pt.y * 2
End Sub

Dim p As Point = (10, 20)
DoublePoint p
Print p.x, p.y    ' Output: 20    40
```

Passing Array Elements

You can pass a single element of a structure array:

```
Dim points(10) As Point
points(5).x = 100
points(5).y = 200

PrintPoint points(5)
DoublePoint points(5)
```

Passing Structure Arrays

Use empty parentheses to pass an entire array of structures:

```
Sub ProcessPoints(pts() As Point)
  ' Access pts(0), pts(1), etc.
End Sub

Dim myPoints(10) As Point
ProcessPoints myPoints()
```

Example:

```
Sub SumAllPoints(pts() As Point, count%)
  Local total_x% = 0, total_y% = 0
  Local i%
  For i% = 0 To count% - 1
    total_x% = total_x% + pts(i%).x
    total_y% = total_y% + pts(i%).y
  Next i%
  Print "Total X:"; total_x%; " Total Y:"; total_y%
End Sub

Dim data(5) As Point
' ... initialize data ...
SumAllPoints data(), 6
```

Functions with Structure Parameters

```
  Function Distance(pt As Point) As FLOAT
    Distance = Sqr(pt.x * pt.x + pt.y * pt.y)
  End Function

  Dim p As Point = (3, 4)
  Print Distance(p)    ' Output: 5
```

Functions Returning Structures

Functions can return structure values using "As typename" in the function declaration:

```
  Function functionname(parameters) As typename
    functionname.member1 = value1
    functionname.member2 = value2
  End Function
```

The function name acts as a local structure variable that is returned when the function exits.

Example - Creating a Point:

```
  Function MakePoint(x%, y%) As Point
    MakePoint.x = x%
    MakePoint.y = y%
  End Function

  Dim p As Point
  p = MakePoint(100, 200)
  Print p.x, p.y    ' Output: 100    200
```

Example - Structure with multiple types:

```
  Function CreatePerson(n$, a%, h!) As Person
    CreatePerson.name = n$
    CreatePerson.age = a%
    CreatePerson.height = h!
  End Function

  Dim employee As Person
  employee = CreatePerson("Alice", 30, 1.68)
  Print employee.name; " is"; employee.age; " years old"
```

Example - Factory function:

```
  Function Origin() As Point
    Origin.x = 0
    Origin.y = 0
  End Function

  Dim startPoint As Point
  startPoint = Origin()
```

Local Structures

Use "LOCAL" to declare structures that exist only within a subroutine or function:

```
  Sub Example
    Local pt As Point
    pt.x = 100
    pt.y = 200
    ' pt is automatically freed when the sub exits
  End Sub
```

Local Structure Arrays

```
  Sub ProcessData
    Local tempPoints(10) As Point
    ' Use tempPoints...
```

```
End Sub
```

Local Structures with Initialization

```
Sub Example
  Local pt As Point = (50, 75)
  Print pt.x, pt.y
End Sub
```

Nested Structures

Structures can contain other structures as members. The nested structure type must be defined before it is used in another structure.

Defining Nested Structures

```
' Define inner structure first
Type Point
  x As INTEGER
  y As INTEGER
End Type

' Now define structure that contains Point
Type Line
  start As Point    ' Nested structure member
  finish As Point   ' Another nested member
  color As INTEGER
End Type
```

Accessing Nested Members

Use chained dot notation to access nested members:

```
Dim myLine As Line
myLine.start.x = 10
myLine.start.y = 20
myLine.finish.x = 100
myLine.finish.y = 200
myLine.color = 255

Print myLine.start.x      ' Prints 10
Print myLine.finish.y     ' Prints 200
```

Multiple Levels of Nesting

Structures can be nested to multiple levels:

```
Type Point
  x As INTEGER
  y As INTEGER
End Type

Type Box
  topLeft As Point
  bottomRight As Point
End Type

Type Scene
  boundary As Box    ' Box contains Points - 3 levels
  name As STRING LENGTH 20
End Type

Dim myScene As Scene
```

```
  myScene.boundary.topLeft.x = 0
  myScene.boundary.bottomRight.x = 640
  myScene.name = "MainScene"

  Print myScene.boundary.topLeft.x     ' Prints 0
```

Arrays with Nested Structures

Arrays of structures containing nested structures work as expected:

```
  Dim lines(10) As Line
  lines(0).start.x = 1
  lines(0).start.y = 2
  lines(5).finish.x = 100
```

Arrays of Nested Structure Members

Structure members can be arrays of nested structures:

```
  Type Point
    x As INTEGER
    y As INTEGER
  End Type

  Type Polygon
    vertices(9) As Point   ' Array of 10 nested Point structures
    color As INTEGER
  End Type

  Dim shape As Polygon
  shape.vertices(0).x = 0
  shape.vertices(0).y = 0
  shape.vertices(1).x = 100
  shape.vertices(1).y = 50
  shape.color = 255
```

Complex Nesting Example

The most complex supported syntax combines all features:

```
  Type InnerType
    values(9) As FLOAT    ' Array of floats
  End Type

  Type OuterType
    items(5) As InnerType  ' Array of nested structs, each with array
  End Type

  Dim data(3) As OuterType  ' Array of outer structs

  ' Access: array(i).array_member(j).array_member(k)
  data(2).items(1).values(4) = 3.14159
  Print data(2).items(1).values(4)   ' Prints 3.14159
```

This demonstrates:

- "data(2)" - Element 2 of the outer array
- ".items(1)" - Element 1 of the nested struct array member
- ".values(4)" - Element 4 of the innermost float array

LIST TYPE with Nested Structures

The "LIST TYPE" command shows nested structure types by name:

```
  >LIST TYPE Line
  TYPE LINE
      START AS Point  ' offset=0
      FINISH AS Point  ' offset=16
      COLOR AS INTEGER  ' offset=32
  END TYPE  ' size=40 bytes
```

Limitations

- The nested type must be defined BEFORE the containing type
- No self-referential structures (a type cannot contain itself)
- Maximum nesting depth: 8 levels (configurable via "MAX_STRUCT_NEST_DEPTH")

Multiple Structure Types

You can define multiple different structure types in your program:

```
  Type Point
    x As INTEGER
    y As INTEGER
  End Type

  Type Rectangle
    left As INTEGER
    top As INTEGER
    width As INTEGER
    height As INTEGER
  End Type

  Type Line
    start As Point    ' Nested structure
    finish As Point
  End Type

  Dim p As Point
  Dim r As Rectangle
  Dim ln As Line
```

Best Practices

1. **Define types at the start of your program** - Place all TYPE definitions near the beginning, before any executable code.
2. **Define nested types before containing types** - Inner structures must be defined first.
3. **Use meaningful names** - Choose descriptive names for both types and members:

```
     Type SensorReading
       timestamp As INTEGER
       temperature As FLOAT
       humidity As FLOAT
     End Type
```

4. **Initialize structures** - Always initialize structure members before use, either with the initialization syntax or by assignment.
5. **Use LOCAL for temporary structures** - When a structure is only needed within a subroutine, declare it as LOCAL to automatically free memory.
6. **Pass structures to subroutines** - Rather than passing many individual parameters, group related data into a structure.

Limitations

- Maximum structure types: 32
- Maximum members per structure: 16
- Member names follow standard MMBasic variable naming rules

- Maximum nesting depth: 8 levels (configurable via "MAX_STRUCT_NEST_DEPTH")

Error Messages

Error	Cause
"Structure t...	The structure type name in DIM AS doesn't match any defined TYPE
"Unknown str...	Accessing a member name that doesn't exist in the structure
"Structure t...	Trying to copy or pass structures of different types
"Expected a ...	A subroutine expected a structure but received something else
"Source must...	STRUCT COPY source is not a structure
"Destination...	STRUCT COPY destination is not a structure
"Not enough ...	Initialization list has fewer values than required
"Expected '(...	Missing opening parenthesis in initialization
"Expected a ...	STRUCT.FIND requires a structure array, not a single variable
"Member not ...	STRUCT.FIND or STRUCT SORT member name doesn't exist
"Cannot sear...	STRUCT.FIND cannot search members that are arrays
"Type mismat...	STRUCT.FIND search value type doesn't match member type
"Type mismat...	STRUCT.FIND search value is not a string but member is
"Expected #f...	STRUCT SAVE/LOAD requires a file number starting with #
"Invalid fil...	File number is outside valid range
"File not op...	STRUCT SAVE/LOAD file is not open
"Not a disk ...	STRUCT SAVE/LOAD requires a disk file, not serial port
"Cannot save...	STRUCT SAVE/LOAD requires whole structure, not member
"Array varia...	STRUCT SAVE/LOAD array must use parentheses

Complete Example

```
' Define structure types
Type Point
  x As INTEGER
  y As INTEGER
End Type

Type Line
  name As STRING LENGTH 20
  startX As INTEGER
  startY As INTEGER
  endX As INTEGER
  endY As INTEGER
End Type

' Declare variables
Dim origin As Point = (0, 0)
Dim cursor As Point
Dim lines(10) As Line

' Initialize cursor
cursor.x = 100
cursor.y = 100

' Create some lines
lines(0).name = "Horizontal"
lines(0).startX = 0 : lines(0).startY = 50
lines(0).endX = 100 : lines(0).endY = 50

lines(1).name = "Vertical"
lines(1).startX = 50 : lines(1).startY = 0
```

```
  lines(1).endX = 50 : lines(1).endY = 100

  ' Subroutine to calculate line length
  Function LineLength(ln As Line) As FLOAT
    Local dx% = ln.endX - ln.startX
    Local dy% = ln.endY - ln.startY
    LineLength = Sqr(dx% * dx% + dy% * dy%)
  End Function

  ' Print line information
  Sub PrintLine(ln As Line)
    Print ln.name; ": ("; ln.startX; ","; ln.startY; ") to ("; ln.endX; ","; ln.endY; ")"
    Print "  Length: "; LineLength(ln)
  End Sub

  ' Display all lines
  For i% = 0 To 1
    PrintLine lines(i%)
  Next i%
```

Output:

```
  Horizontal: (0,50) to (100,50)
    Length: 100
  Vertical: (50,0) to (50,100)
    Length: 100
```

Quick Reference

Commands

Command	Description
`Type...End ...	Define a new structure type
`Dim var As ...	Declare a structure variable
`Dim arr(n) ...	Declare an array of structures
`dst = src`	Copy structure using assignment
`arr(i) = ar...	Copy array elements using assignment
`Struct Copy...	Copy structure contents
`Struct Copy...	Copy entire structure array
`Struct Sort...	Sort array by member field
`Struct Clea...	Reset all members to defaults
`Struct Clea...	Reset all array elements to defaults
`Struct Swap...	Exchange contents of two structures
`Struct Prin...	Print structure contents for debugging
`Struct Prin...	Print all array elements
`Struct Save...	Save structure to open file
`Struct Save...	Save entire structure array to open file
`Struct Save...	Save single array element to open file
`Struct Load...	Load structure from open file
`Struct Load...	Load entire structure array from open file
`Struct Load...	Load single array element from open file

Functions

Function	Description
`Struct(FIND...	Find element with matching member value, returns index or -1
`Struct(SIZE...	Get upper bound of structure array dimension

`Struct(SIZE...	Get size in bytes of a structure type

Member Types

Type	Size	Alignment	Description
`INTEGER`	8 bytes	8-byte	64-bit signed integer
`FLOAT`	8 bytes	8-byte	64-bit floating point
`STRING`	256 bytes	None	Default string (255 cha...
`STRING LENGTH n`	n+1 bytes	None	String with specified m...
Nested struct	Varies	8-byte	Size of the nested stru...