Collections and class lists

AppleScript does not have collection objects. Instead, you use the plural form of any class, which is a list. The dictionary always provides the plural if there is one. On very rare occasions, there is no plural, and then you can't get a list.
The only classes that don't have a plural are those in which there is only one object, such as **application** and **selection**. In all other cases, a class is an element of another class — as noted in the "contained by" sections of dictionary entries — or of the application itself.

The every [class name] syntax

There is a commonly used, preferred synonym for the plural: **every***[class name]*. The usual `For Each` loop in Visual Basic for Applications (VBA) that iterates through every member of a collection is replaced by a `repeat` loop in AppleScript, as shown in the following statements:

- `repeat with someObject in (every element)`
- `repeat with i from 1 to (count every element)`

To get an idea of these constructions in action, compare the following examples used to unlink fields:

VBA

```
For Each oField In ActiveDocument.Fields
        oField.Unlink
Next oField
```

AppleScript

```
tell application "Microsoft Word"
        set allFields to (every field of active document)
        repeat with oField in allFields
                unlink oField
        end repeat
end tell
```

The VBA macro operates on the **Fields** collection, while the AppleScript example uses the **every field** syntax.

Commands often can't be used on lists

Because commands can act on a whole list at once in AppleScript, you might presume that you can avoid the `repeat` loop by using the plural in the manner shown in the following example:

```
tell application "Microsoft Word"
        unlink (every field of active document)
end tell
```

However, using a command with a list does not always work.
The example above returns an error, even if you use the explicit **get** or a variable. The error explains that `{field 1 of active document, field 2 of active document,...}` doesn't understand the unlink message. If you check the document, you'll see that the first field was in fact unlinked, but none of the others were. The reason is simply that the **unlink** command was not implemented to work on lists.
In most cases in Microsoft Word, Excel, and PowerPoint, you cannot use commands with lists. However, that is not true in Entourage. For the most part, commands mirror the VBA methods from which they were derived, and methods don't use lists. Therefore, you usually have to use `repeat` loops with lists.

Assign plurals to a variable when using repeat loops

As shown in the AppleScript example above that uses a `repeat` loop, you have to set a variable and refer to it, like `allFields` in the example.
If you try the following code instead, with no variable, the code returns an error, even when using the explicit **get**:

```
repeat with oField in (get every field of active document)
        unlink oField
end repeat
```

The error explains that **field** does not understand the unlink message. That's because the fields keep getting re-indexed as they get unlinked, skipping every second one, until the index is larger than the number of fields left. Instead, if you don't use a variable, you have to use the following code:

```
tell application "Microsoft Word"
        repeat with i from (count (get every field
                of active document)) to 1 by -1
                set oField to item i of (get every field
                        of active document)
                unlink oField
        end repeat
end tell
```

The example above requires the following elements:

- The explicit **get**
- The `repeat with i from` loop
- A count backwards from the last item to the first by –1

When you don't use a variable, you have to `get every field of active document` twice, sending an AppleEvent each time. Because the AppleEvents are unnecessary and slow, use a variable instead, like the example that uses `allFields`, above.
You can also use a `repeat with i` loop, as shown in the following example:

```
tell application "Microsoft Word"
        set allFields to (every field of active document)
        repeat with i from 1 to (count allFields)
```

```
                set oField to item i of allFields
                unlink oField
        end repeat
end tell
```

Using classes that are elements of other classes

The example below is a VBA macro that unlinks every header and footer in the document. This example provides several opportunities to examine differences in objects between VBA and AppleScript.

```
Dim oField As Field
Dim oSection As Section
Dim oHeader As HeaderFooter
Dim oFooter As HeaderFooter

For Each oSection In ActiveDocument.Sections

    For Each oHeader In oSection.Headers
        If oHeader.Exists Then
            For Each oField In oHeader.Range.Fields
                oField.Unlink
            Next oField
        End If
    Next oHeader

    For Each oFooter In oSection.Footers
        If oFooter.Exists Then
            For Each oField In oFooter.Range.Fields
                oField.Unlink
            Next oField
        End If
    Next oFooter

Next oSection
```

This example unlinks fields in the **Ranges** of headers and footers, rather than in the main body of the document. Interestingly, the code is completely different in AppleScript for the following reasons:

- There are no collection objects
- You cannot create a list of headers and footers

Because the **header footer** class is not an element of any other class, not even of **section**, as you might expect, you can't create a list for it. These obstacles make the task more difficult in AppleScript.

The reason that **header footer** isn't an element of another class is likely because of the AppleScript rule that you cannot have read-only elements (you can have read-only properties). An object can have 0 to an infinite number of elements, and you only have to use `make new element at someObject` to create a new one.

Meanwhile, the object models for Word, Excel, and PowerPoint contain many collection objects that have no **Add** method; you can only use what you're given. The **HeaderFooters** collection object of each document **Section** is one of these types. You can use, at a maximum, three headers and three footers: the *primary*, *first page*, and *even pages* headers or footers. You cannot add your own.

Therefore, in AppleScript, **header footer** objects cannot be elements of **section** because you cannot make an unlimited number of them. They have to be self-standing objects that are divorced from the object model. The dictionary says they are inherited from **base object**.

Use proprietary commands with these classes

To get these objects, you use the **get header** and **get footer** proprietary commands, rather than, for example, a statement like this: `header footers of section 1 of active document`.

You also have to use an **index** parameter to specify which header or footer you want. This construction is similar to using an index with the **Headers** and **Footers** properties of **Section** in VBA, as shown in the following example:

```
ActiveDocument.Sections(1).Headers(wdHeaderFooterPrimary)
```

The equivalent code in AppleScript is as follows:

```
get header (section 1 of active document) index
    header footer primary
```

However, there's no easy way to get every header or every footer. There is no code in AppleScript that is the equivalent of `ActiveDocument.Sections(1).Headers` in the VBA example above.

Instead, you have to get each header and footer by index. You have a couple of options:

- Run each one through a handler to get its text object (**range**) and fields.
- Put each into a single-item list framed by list braces, concatenate these lists into one list, and then run a `repeat` loop on the list.

The following code executes the second option:

```
tell application "Microsoft Word"
        set theSections to every section
            of active document
    repeat with theSection in theSections
        set theHeaderFooters to
            {get header theSection index
            header footer primary} &
            {get header theSection index
            header footer first page} &
            {get header theSection index
            header footer even pages} &
            {get footer theSection index
            header footer primary} &
            {get footer theSection index
            header footer first page} &
            {get footer theSection index
            header footer even pages}
```

```
                    repeat with theHeaderFooter in theHeaderFooters
                            set theFields to every field
                                    of text object of theHeaderFooter
                            repeat with theField in theFields
                                    unlink theField
                            end repeat
                    end repeat
            end repeat
    end tell
```

Look out for other classes that are not elements of another class and are only available using proprietary commands. There are quite a few of these around.

Don't use exists tests with these classes

One thing you may notice is that you don't have to run an `exists` test on the **header footers** as you do in VBA. In the VBA model, only primary headers and footers exist by default, even if they are empty. So, you have to check whether *first page* and *even pages* varieties exist, which are the constants *wdHeaderFooterFirstPage* and *wdHeaderFooterEvenPages*.
In AppleScript, because the commands **get header** and **get footer** can apply the parameters *header footer first page* and *even pages* and get a result, all of them exist by default, even if empty. There are no problems with empty lists for `theFields` when the `text object of theHeaderFooter` has none.