Objects and collections

There are differences between the AppleScript and Visual Basic for Applications (VBA) object models for the Microsoft Office applications.

No collection objects

One of the most important differences is that there are no collection objects in AppleScript. Unlike VBA, where single collection objects incorporate all existing instances of their singular "client" objects, in AppleScript, there are only singular object classes in the dictionary.
If there are several object instances, either as elements of the application or as elements of some other class, they can be considered together in a list, which is like a VBA array. There is no special treatment for lists of application objects as opposed to, say, a list of integers {1, 2, 3, 4, 7, 97, 1456863}. It's just a list.
Classes have a plural form in the dictionary. For most classes, it is **every text range**. Plural forms also have a synonym, such as **text ranges**. You can usually send commands to act on the whole list at once.
Although there are no collection objects in AppleScript, the plural form of each element does constitute a collection. The plural form is the list of all instances of that element of any class, or of the application. So, the method for referring to a particular element in AppleScript is similar to the implicit **Item** method of VBA collection objects, but without the parentheses.

Referring to a single element

Most elements can be referred to by index, such as `document 2`, or `paragraph 1 of active document`. You do not have to explicitly specify when an item is an element of an application, like `document 2`, above. The index can change under certain conditions, of course, such as when elements with a lower index are deleted.
There are also alternative constructions. If a class has a **name** property, you can generally specify it by name by following the element's class with the name string, such as `document "Converting VBA to AppleScript.doc"`, or `document property "Author" of active document`.
Sometimes, with built-in elements like Microsoft Word styles, or for applications that maintain databases like Microsoft Entourage, you can specify elements by unique ID.
You can also use `whose` filters, available to almost all elements in the Office applications. For more information about `whose` filters, see Other language differences.

Creating new objects

So how do you add a new instance of a class without a collection object to add it to? The answer is that you don't. You usually create a new object with the **make new** command from the Standard Suite, as shown in the following example:

```
make v : Make a new element

make
        new type class : the class of the new element.
        at location reference : the location at which to
                insert the element
        [with data anything] : the initial data for the element
        [with properties record] : the initial values for the
                properties of the element
        -->reference : to the new object(s)
```

The almost universal way to make a new class instance is to `make new [element] at [a location] with properties {...record...}`. The instance can be an existing element of the application or of some other class. An AppleScript record is an unordered collection of key-value pairs.
If you make a new object that is listed as an element of the **application** itself—for example, if you are working in Word and the new object is listed in the **application** class entry in the Microsoft Word Suite—omit the **at** parameter, as is the case with `new document` in the example below. Compare the following examples:

VBA

```
Documents.Add
```

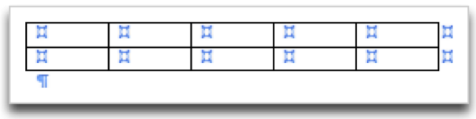
AppleScript

```
tell application "Microsoft Word"
        make new document
end tell
```

Default values for properties

In AppleScript, any property that you don't specify with an initial value in the **with properties***{...}* parameter already has a value by default. It is possible to leave them all unspecified, as with `new document`, above, and in the following example:

```
make new table at active document
```



Note To save space, some of the `tell` blocks to `"Microsoft Word"` have been removed, but they are required in your script.
The result of this script, as illustrated above, is a default table at the cursor with two rows and five columns. This isn't the result with VBA's **Tables.Add** method, in which you must specify the required arguments: **Range**, **NumRows**, and **NumColumns**. However, in AppleScript, there are default values for all properties, so you

can use **make new** without them and you won't receive an error.

Specifying properties

Most of the time, of course, you will be specifying properties at creation time using **with properties**, as shown in the following example:

```
tell application "Microsoft Word"
        set myDoc to make new document
        set myTable to make new table at myDoc with properties
                {allow auto fit:true, allow page breaks:true,
                        column options:
                        {default width:1.0, preferred width:25,
                                preferred width type:
                                preferred width percent},
                        number of columns:4, number of rows:12,
                                spacing:2.0}
end tell
```

This example sets some **table** properties at inception. There are a lot more available, as a glance at the dictionary entry for **table** shows. In fact, you can set the same properties as those in VBA, with very similar nomenclature. Properties that are not specified have default values.
Neither the dictionary nor the Word AppleScript Reference tell you what the property defaults are for **table**, which is also true for **Table** object in VBA Help. In both languages, in cases like these, you have to test them out.
Note that the **number of columns** and **number of rows** properties allow you to set these table dimensions at inception, even though the dictionary discloses that they are read-only (r/o) properties. You can only set them in a `make new table` statement, not later in your script. Also interesting is that, in VBA, these properties correspond to the **NumColumns** and **NumRows** parameters of the **Tables.Add** method, rather than to any property of **Table** object.
After the table exists, you can add new elements, including rows and columns, using **at**, as shown in the example below. You can also delete indexed rows or columns.

```
make new row at end of table 1 of active document
```

You can specify the following locations (insertion points) for adding elements:

- `at the beginning`
- `at the end`
- `after` an indexed element
- `before` an indexed element

The following example uses the `after` location:

```
make new row at after row 5 of table 1 of active document
```

Pitfalls to watch out for

You can use the `make new [object] with properties` statement with most classes in Word, Excel, and PowerPoint, as well as every other scriptable application on the Macintosh. Occasionally, however, you can get a bug using this construction with the Office applications because of an element of the OLE Automation / VBA heritage that couldn't be adapted to standard AppleScript usage.
In these cases, you can use "proprietary" commands, which usually begin with the word **create** rather than **make**. For example, in Word, there are seven commands in the Microsoft Word Suite that begin with **create**. The **create range** command is a frequently used example.
Look for **create** and other proprietary commands if you run into any **make new** errors. The good news for veteran VBA macro writers is that the **create** commands correspond closely to VBA methods.
When you are using the **with properties** parameter, there is an exception in how it is used when you are making a new Word document. You cannot specify the document's properties at inception because they will be ignored. You must set them immediately after making the document, as you do with the **Add** method in VBA. This exception does not apply to Excel or PowerPoint.
Lastly, in VBA, some collection objects are read-only and you cannot make new members. In AppleScript, these collection objects have not been implemented as elements because there are no "read-only elements" (as opposed to read-only properties) in AppleScript. You can always make new elements.
Instead, the objects, such as **headers** and **footers** in Word, are "stand-alone" classes that can only be referenced using proprietary commands in the dictionary, not as elements or properties of a containing class, such as **section**. Always check the dictionary to verify if an object is "contained by" a class other than **base** object, or if you can only get it by a specific command that you might need to search for using the dictionary's Search box.