

# Keeping the Memory Clean - Part 1

By Walt Nelson, Director of Partner Development

4<sup>th</sup> Dimension Technical Note 97-32

Technical Notes for 97-09-September 1997

## Introduction

---

At any given time, the performance and stability of any 4D application is affected by the amount of RAM that is available for loading objects into memory, and by the sizes of the blocks of available RAM.

In this two-part technical note, you will learn how 4D uses memory, and what you can do in order to keep the memory clean. In Part 1, we discuss how 4D uses memory, and offer some tips for optimizing the memory in your databases and forms. In Part 2, you will learn how to write 4D code that helps to keep the memory clean.

## How 4D Uses Memory

---

In this technical note, when we refer to 4D, this includes 4th Dimension, 4D<sup>®</sup>Runtime, 4D Engine, 4D Client, and 4D Server.

In this discussion, we will use the default memory settings. You can use 4D<sup>®</sup>Customizer Plus (version 3 and version 6) or the Database Properties dialog (version 6) to change the defaults.

When it launches, 4D divides the available memory into two main categories: kernel memory and data cache.

### Kernel memory

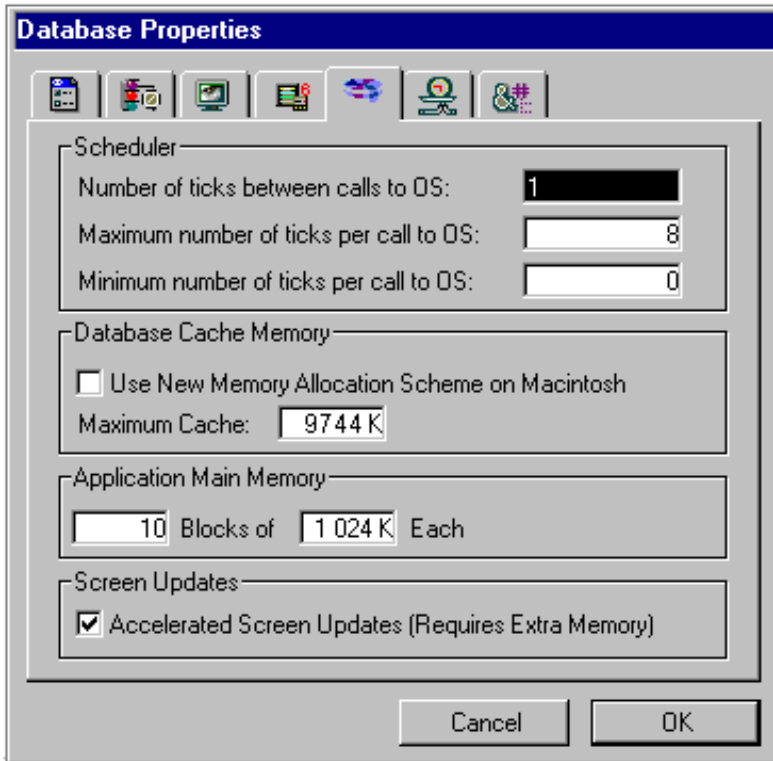
This is RAM that 4D uses for process stack space, forms, methods, plug-ins, sets, variables, current records, current selections, and named selections. In client/server systems, some of these objects are stored on the client and some are stored on the server. However, since the objects originate on the server and are then sent to the client, the server will have to load every object—even if only for a brief period of time!. As the program runs, 4D swaps kernel segments in and out of this memory.

### Data cache

The next block of memory reserved by 4D is the data cache. The size of the data cache determines how many address tables, bit tables, records, and indexes can be kept in memory for faster searches, sorts, and record retrieval operations. When 4D needs a data object, it first looks in the data cache. If the object is there, 4D can load it several times faster than it can load the same object from disk. If the object is not already in the data cache, it is automatically added to the cache and, if necessary, the oldest objects in the cache are purged in order to make room for this new object.

The Customizer Plus settings for the data cache on Windows and Macintosh are different, because of differences in how the two operating systems manage memory. On the Macintosh, the default setting for the data cache is 50% of the memory remaining after the kernel memory has been reserved. For example, if you are running 4D Server with 20,000k and the kernel memory is 512k, then you have 19,488k remaining. Of this, 50% (9,744k) is assigned to the data cache. On Windows, the data cache is a specified amount of RAM. You can use 4D Customizer Plus (version 3 and version 6) or Database Properties (version 6) to specify the size of the data cache.

Let's assume, for this discussion, that you set the data cache to 9744k as shown.



**Note:** The data cache does not apply to 4D Client, because 4D Client does not store any data. In client/server systems, all data is stored on the server, so there is no need for a data cache on the client.

### What is Clean Memory?

---

When an object is loaded into memory, 4D needs to have one contiguous block of memory in which to load the object. For example, if the object is a several-page form that is 350k in size, then 4D needs a single block of RAM that is at least 350k in order to load that form. This is an important point: even if 4D has more than 350k of memory available, 4D cannot load a large object if the memory is fragmented. In such a case, 4D will have to "purge and merge," that is, remove unused objects from memory, and then move the remaining objects around to create a block large enough to load the object. At best, there will be a delay; at worst, 4D will not be able to create a block large enough to load the object. Obviously, then, the ideal situation is to keep the size of your objects small, and to keep the size of the available memory blocks large. When you have this combination, you have "clean memory."

### Memory-Efficient Form Design

---

When you design your forms, you should always keep in mind the memory consequences of your form design. Here are some examples of things to consider:

#### Size of graphics

Most forms will contain graphical objects such as pictures and buttons. Make sure you are aware of the size in bytes of every graphical object on your forms. Suppose, for example, you have an output form containing a logo that is 20k in size, and seven of your own custom buttons (New record, Delete records, Report, Graph, etc.). Let's further assume that the average size of a button is 6k. This means that your form is using 20k plus 42k, or 62k for graphics.

#### Number of pages

Continuing with our example, let's assume that your form has five pages, and each page contains those same custom graphics. So, the total size of the custom graphics on your form is: 5 pages \* 62k = 310k.

### **Number of processes**

Let's assume that your application allows the user to open several processes and display input and output forms in each process. If the user has three processes open, and they are all displaying output forms, that's 3 \* 310k = 930k, just to display graphics on output forms!

### **Size and number of methods on forms**

When 4D loads a form, it also loads the form method, the object methods, and all project methods that are called in the On Load event (the equivalent of the Before phase in version 3). Let's assume that an input form contains 10 object methods that are each 2k in size; the form also contains a form method that is 4k in size; and in the On Load event, the form method calls two project methods that are each 4k in size. This means that this form will load methods that total 20k + 4k + 8k = 32k.

### **Number of variables**

If you have a large number of variables on a form, the form will use quite a bit of memory and will be slow to load. If your form has more than 20 or 30 variables, start looking for ways to optimize.

## **General Form Design Recommendations**

---

Based on our observations, here are some tips to help you to keep the memory clean with regard to your 4D forms.

### **Limit the color depth of graphics**

If you are going to use custom graphics on your forms, you need to make sure that those graphics use as little memory as possible. Pay close attention to color depth when designing your graphics: if you only need 256 colors (8-bit graphics), make sure that your screen depth is not set to thousands or millions of colors (16-bit or 32-bit). Also, as an alternative to custom buttons, consider using the button graphics supplied in 4D V6. Each of these buttons is less than 1k in size: most are between 600 bytes and 800 bytes.

### **Use the V6 picture library**

If your application is in 4D V6 and you have custom graphics that you want to display in your forms, store those graphics in the picture library. When you do this, 4D only needs to load one instance of the picture or variable.

Here is an example of a picture library that contains custom buttons and other custom graphical objects.

### Version 3: Store standard pictures in a [Picture\_Objects] table

In version 3, you can store graphics as fonts, as resources, or as data that is loaded into interprocess variables on startup. Since application-installed fonts are not supported on Windows and Resources are not directly supported on Windows, you should probably use the data method if your applications will be cross-platform. For example, suppose you need to display your own custom OK and Cancel buttons. Here are the steps for storing those objects as data:

1. In your structure, create a table called [Picture\_Objects].
2. Create two fields: [Picture\_Objects]Object\_Name and [Picture\_Objects]Picture.
3. Create several records in this table: one record for each graphical object to be stored. In these records, you will store the name of the object (e.g., Button\_OK) and the graphic of the object.
4. In your Startup method, copy each of these graphics into an interprocess variable. The procedure might look something like this:

```
`Global procedure: Startup
C_PICTURE(<>bOK;<>bCancel)
ALL RECORDS([Picture_Objects])
For ($Records;1;Records in Selection([Picture_Objects])
  Case of
    :([Picture_Objects]Object_Name=ÔButton_OKÓ)
      <>bOK:= [Picture_Objects]Picture
    :([Picture_Objects]Object_Name=ÔButton_CancelÓ)
      <>bCancel:= [Picture_Objects]Picture
  `Add a case for each graphic.
  End case
  NEXT RECORD([Picture_Objects])
End for
```

5. Place these interprocess variables on your forms. No matter how many forms display the graphics at one time, there will only be one instance of the variable in memory.

### Do not put too many lines of code in your object methods

As mentioned earlier, when 4D loads a form, it also loads the form method, the object methods, and all project methods that are called in the On Load event (the equivalent of the Before phase of version 3). The problem with putting code into object methods (scripts) is that there are usually several of them on the form--especially on an input form. Even if each object method is only 1k or 2k, the cumulative effect can be significant. Therefore, instead of putting many lines of code in

your object methods, put the code into project methods and make one-line calls to these project methods (equivalent to global procedures in version 3). This means that 4D will not use memory to store code that is not yet needed. It will call the method when needed, use it, and then free that memory for re-use.

### **Use arrays instead of a large number of individual variables**

Whenever possible, avoid using more than a few variables—find other ways to do what needs to be done. For example, it is far better to display ten 20-element arrays than to display 200 individual variables.

### **Summary**

---

At any given time, the performance and stability of any 4D application is affected by the amount of RAM available for loading objects into memory, and by the sizes of the blocks of available RAM. In this technical note, you have learned how 4D uses memory, and what you can do in your 4D forms to keep the memory “clean.” In Part 2, you will learn how to write 4D code that helps to keep the memory clean.