

# Introducing S-PLUS 6

*David M Smith, S-PLUS Product Marketing Manager*

Version: 1.1

*In mid-2001, Insightful released S-PLUS 6 for Windows following the late 2000 release of UNIX platforms. This is a significant event in S-PLUS history, being the first time since S-PLUS 3 that the Windows and UNIX versions have been released from the same code base. This also reflects the first time that the Windows and UNIX versions have been truly cross-compatible: S-PLUS 6 for Windows and UNIX will both include version 4 of the next-generation S language engine from Lucent Technologies.*

*In this white paper, we outline the goals and achievements of the S-PLUS 6 release in the context of the history of S and S-PLUS. Discussion of new features available in S-PLUS 6 for UNIX and Windows is included.*

## 1. Background: A brief history of S and S-PLUS

In the late 1970s, researchers Rick Becker, John Chambers, Doug Dunn, Paul Tukey, and Graham Wilkinson at AT&T Bell Labs (now Lucent Technologies) set about designing and implementing a system to facilitate their data analysis requirements through software. Drawing on their various expertise and the then-current trends in exploratory data analysis, they implemented the first version of **S**<sup>1</sup>: a programming language designed expressly for data analysis and statistical graphics.

The S system created an enthusiastic following through its use within Bell Labs and in academic research. Seeing potential for more widespread use, Doug Martin – a professor at the University of Washington in Seattle – started Statistical Sciences Inc. in 1987 with an exclusive license from Bell Labs to commercially develop S. By taking the S engine from Bell Labs and adding documentation, support and extra functionality, **S-PLUS**<sup>2</sup> was born. In 1993, Statistical Sciences was sold to MathSoft (since renamed Insightful Corporation), and today S-PLUS is produced and marketed by Insightful from its offices in Seattle, Washington.

Since S-PLUS was first made available, both S and S-PLUS have continued to evolve. S has progressed through several versions introducing features such as object-orientation, statistical modeling and interfaces with other languages. The advances made here were recognized in 1998, when the S language was awarded the prestigious ACM Software Systems Award<sup>3</sup> for having "forever altered how people analyze, visualize and manipulate data". Successive versions of S-PLUS from Statistical Sciences and (latterly) Insightful have incorporated the latest version of S from Bell Labs, as well as introducing independently developed features such as additional statistical libraries and graphics devices, an interactive graphical user interface (GUI) on Windows, and interoperability with numerous files, applications and operating systems.

---

<sup>1</sup> See <http://cm.bell-labs.com/cm/ms/departments/sia/S/index.html> for more information about the history and evolution of the S language.

<sup>2</sup> More information about S-PLUS is available at from [www.insightful.com/products/desktop.asp](http://www.insightful.com/products/desktop.asp).

<sup>3</sup> See <http://www.acm.org/awards/ssaward.html> for a history of ACM Software System awards.

Today, S-PLUS has more than 100,000 users worldwide in a variety of areas including academic research, finance, biotechnology, manufacturing and government.

## 2. S version 4 and S-PLUS 6

John Chambers continued the evolution of the S system to produce S version 4 (Sv4), a major upgrade of the S system. It provides a more explicit object-oriented class system than previous versions of the S language, as well as facilities to handle large data sets, and integrate better with C applications. That work culminated in the publication of his book *Programming with Data* (Springer, 1998) and the delivery of S version 4 to Insightful in 1997. S version 4 was ported to the various UNIX platforms supported by Insightful and incorporated into S-PLUS 5 for UNIX and Linux in 1998.

S version 4 was not made available on the Windows platform at that time; the then-current release of S-PLUS for Windows, S-PLUS 2000, was still based on the S version 3 (Sv3) language engine. The Insightful development team has been hard at work since the Sv4 release to port it to Windows, and to integrate Sv4 with the Windows GUI. That work is now complete, and the current release of S-PLUS for Windows – S-PLUS 6 – features the Sv4 language engine.

Insightful is committed to providing the Sv4 engine in the S-PLUS products on UNIX and Windows for the following reasons:

1. The Sv4 language provides a more complete object-oriented model than Sv3, allowing the development of sophisticated applications based on well-defined classes and methods.
2. The Sv4 language engine includes efficiency improvements such as memory mapping and reference counting which will make programs run faster and scale better to large datasets.
3. Sv4 provides features which allow users to develop applications which can scale to the requirements of large datasets. These features include direct access to binary files and pipes, and the ability to read a dataset in small chunks (block-reads) meaning that users can write algorithms where computations are done on a dataset without the requirement of the entire dataset being in memory at once.
4. Insightful is dedicated to continuing its longterm commitment to Lucent Technologies to deliver the latest innovations to the S engine in S-PLUS.

In addition to introducing the Sv4 language engine, S-PLUS 6 for Windows and UNIX realizes a number of goals. They are:

1. To provide cross-platform compatibility between the UNIX and Windows versions of S-PLUS, through the integration of Sv4 engine into S-PLUS for Windows, and the consolidation of the code base for both versions.
2. To provide new capabilities and statistical techniques requested by users.
3. To improve the performance of the Sv4 engine, both in terms of speed and memory consumption.
4. To improve the quality of S-PLUS by fixing bugs reported in the GUI and the engine.

More information about these goals and their impact in S-PLUS 6 is provided in the following sections.

### 3. S version 4 on UNIX and Windows

S-PLUS 6 represents the reunification of the UNIX and Windows versions of S-PLUS. For the first time since S-PLUS 3, both the Windows and UNIX versions of S-PLUS are based on the same underlying language engine: Sv4. In fact, both versions are now built from the same code base to ensure cross-platform compatibility. Furthermore, the UNIX versions of S-PLUS are able to read and write databases created by the Windows version (with files conforming to the limitations of Windows filesystems), and the Windows version is able to read and write databases created by the UNIX versions. This means that the Windows and UNIX versions of S-PLUS can share databases across networks, or collaborate on projects by sharing objects and files in S-PLUS 6.

Sv4 has been available on UNIX and Linux since 1998 as part of S-PLUS 5, but for Windows users the Sv4 language is new to the platform. Naturally, S-PLUS 2000 users who have developed applications using Sv3 will want to continue to use scripts, objects and functions in S-PLUS 6 without difficulty. S-PLUS 6 has been designed so that the transition from S-PLUS 2000 will be as easy as possible. This is achieved in the following ways:

1. By exploiting the largely backward-compatible nature of Sv4 versus Sv3.
2. By providing information about backward-compatibility issues that do exist.
3. By ensuring that S-PLUS 6 can work with existing Sv3 databases.
4. By providing tools to migrate to S-PLUS 6 Sv3 code that is not forward compatible with Sv4.

These are discussed in the following sections.

#### Backward compatibility of Sv4

Although Sv4 provides a major overhaul of the internals of the S language engine, it was designed to be as backward-compatible as possible with Sv3. Function calls, expressions and programming constructs such as loops and conditionals all operate in exactly the same way in Sv4 as they did in Sv3. Operations on the basic data types such as vectors, matrix, list and data frame are identical.

As a result, the vast majority of command-line users will notice no difference switching from S-PLUS 2000 to S-PLUS 6 on Windows. Scripts written for S-PLUS 2000 will generally run without modification in S-PLUS 6.

Even though backwards compatibility with Sv3 was a primary concern of the design of Sv4, there are still a couple of areas where Sv3 code will require modification to run correctly in Sv4. These areas are largely confined to the class system and links with other languages, so if in writing functions and scripts:

- You never created your own classes or methods
- You never called your own C or Fortran code

you should be able to continue using those scripts in S-PLUS 6 without modification. In addition, use-written help files created for use with S-PLUS 2000 will need to be migrated to the new S-PLUS 6 help system. Details of such areas of change are provided with the S-PLUS 6 documentation.

## Migrating from S-PLUS 2000 to S-PLUS 6

While the changes required to port Sv3 applications to Sv4 are generally not excessive, we understand the necessity for help in this area. We have learned many lessons from the transition from Sv3 to Sv4, when S-PLUS 5.0 was introduced to replace S-PLUS 3.4 on UNIX systems:

- Many system managers installed S-PLUS 5.0 as a replacement to S-PLUS 3.4, but many users indicated that they would have liked to have continued to use S-PLUS 3.4 until they were ready to transfer to S-PLUS 5. In response to this feedback we have ensured that S-PLUS 6 will install as a separate application to S-PLUS 2000; it does not replace S-PLUS 2000. You can continue to have S-PLUS 2000 installed until your migration to S-PLUS 6 is complete, and use both applications independently.
- S-PLUS 5.0 for UNIX could be started using an S-PLUS 3.4 database as the working database. This would work (the Sv4 engine can read Sv3 objects), but it would leave the database unusable by S-PLUS 3.4, and you were unable to return to using S-PLUS 3.4 with that database (this problem was fixed with S-PLUS 5.1). To avoid this situation on Windows, S-PLUS 6 will automatically detect if it is started with an S-PLUS 2000 database as its working database, and provide assistance with migrating Sv3 objects to a new database if requested. The original database will not be modified in any way, and it will be possible to continue using S-PLUS 2000 with that database.
- S-PLUS 5 for UNIX provided a tool (`convertOldLibrary`) for migrating an existing Sv3 database for use with Sv4. Feedback from users indicated that this was not sufficient for migration purposes, particularly for those users who kept most of their applications in the form of script files rather than as S objects. S-PLUS 6 for Windows provides a dedicated tool – the Migration Wizard – which will allow users to migrate existing projects for use with S-PLUS 6. In addition to transferring and migrating objects, it will also scan existing script files for potential incompatibilities with Sv4, and allow suitably modified versions to be saved into a new project directory. The goal of this tool is to make it as easy as possible to continue your existing S-PLUS 2000 based work with S-PLUS 6. A command-line tool (`CONVERTOLDSCRIPTS`) is also provided for UNIX users to migrate Sv3 scripts for use with S-PLUS 6.

In summary, we believe that Sv4 is a major milestone for S-PLUS, and that the changes will provide great benefits for all S-PLUS users. Difficulties in transition from Sv3 to Sv4 are inevitable, but by providing support in S-PLUS 6 to make the change we hope to make the transition as straightforward as possible.

## 4. New features in S-PLUS 6 for Windows and UNIX

S-PLUS 6 includes many new features on both the UNIX and Windows platforms. Some features previously available only on Windows or only on UNIX are now available on both platforms. Also, in response to customer requests a number of exciting new features have been implemented on either the UNIX or Windows version of S-PLUS.

### New features common to both UNIX and Windows

- **Java “graphlets”**, a new type of graphics device where the output is an independent Java applet instead of a flat graphics file. This applet can be embedded into a Web-page, and allows the graph to be scrolled and zoomed without loss of resolution. Graphlets also support interactive features such as being able to link components of the graph to other Webpages.

- **The Robust Statistics library**, providing robust techniques in regression, analysis of covariance, and time series analysis.
- **The Missing Data library**, providing multiple imputation techniques based on Gaussian, Logistic and Conditional Gaussian models.
- **Improved file import and export capabilities**, including MatLab 5 and SAS 7 and 8.
- **Windows Metafile graphics device** allowing the creation of WMF documents, so that graphics created in either Windows or UNIX can easily be inserted into Microsoft Office compatible documents.

## New features in S-PLUS 6 for Windows

- **Integration with Microsoft Excel.** In S-PLUS 6, you are able to open an Excel workbook as a document inside S-PLUS. This allows you to use all of the usual Excel features as a data editor for S-PLUS objects. In addition, you can select regions from an existing Excel worksheet and use the data thus selected with the standard S-PLUS plotting tools and statistics dialogs.
- **CONNECT/C++.** This suite of classes and methods for handling S-PLUS objects in C++ allows you to work with S-PLUS objects such as vectors, arrays, lists and data frames directly in C++, and to evaluate arbitrary S-PLUS expressions and have the results returned to C++. You can also create an independent C++ application which makes use of S-PLUS analytics.
- **The Migration Wizard**, which allows you to migrate existing objects and script files from S-PLUS 2000 for use with S-PLUS 6.
- **A version update tool**, allowing you to automatically check for and download the latest release of S-PLUS over the Internet.
- **A new time series object library**, with complete support of dates, times, business days, holidays, aggregation and much more.
- **Improved postscript() driver:** the postscript() driver from UNIX has now been ported to Windows.

## New features in S-PLUS 6 for UNIX

- **A graphical user interface** based on Java. It provides an interactive command-line editor, data viewer, report window, menus and dialogs for many commonly-used data analysis and Trellis graphics functions. The TTY-based interface is still available.
- **CONNECT/Java.** This feature gives you the ability to call Java from S-PLUS, and the ability to call S-PLUS functions from Java. You can even create independent Java applications which make use of S-PLUS's analytic or graphical capabilities.
- **A new Java-based graphics device** which provides multiple tabbed pages, improved colormap control and bitmap rendering. This device can be used independently of the Java GUI.
- **Updated analysis capabilities** in survival analysis, nonlinear mixed-effects, discriminant analysis and QC charts bringing the UNIX version in line with S-PLUS for Windows.
- **New documentation system** based on Sun's JavaHelp browser, and updated online documentation.
- **New graphics export capabilities** to commonly-used formats such as JPEG, TIFF, PNG, PNM and BMP.

Further information about the new features in S-PLUS 6 may be found on the S-PLUS Web site at [www.insightful.com](http://www.insightful.com).

## 5. Performance and memory usage

When S-PLUS 5.0 for UNIX was released in 1998 it was the first time the new Sv4 engine from Lucent Technologies had been included in any version of S-PLUS. The new engine promised improved performance compared to the previous version of S-PLUS 3.4, but for some examples in practice this proved not to be the case. The radical change in the underlying engine introduced a number of unforeseen problems to the S-PLUS 5 release.

The Insightful development team had added a number of efficiency improvements to the Sv3 engine from Lucent for inclusion with S-PLUS 3.4. These included features such as loop compaction (to reclaim no longer needed memory with each iteration of a loop) and general efficiency improvements to a number of functions. These improvements were not compatible with the Sv4 engine as delivered, and so were not included in S-PLUS 5.0. This meant that functions (and in particular, functions including `for` loops) which ran successfully in S-PLUS 3.4 could consume a great deal of memory in S-PLUS 5.0 and therefore run very slowly or fail to complete. Some improvements were made in S-PLUS 5.1, but in general there still existed a class of functions which would still run faster in S-PLUS 3.4 than in S-PLUS 5.1. (Equally, there were a number of applications, particularly those involving large data sets, which would run faster in S-PLUS 5.1 than in 3.4.)

There were many claims that S-PLUS 5 was slower than S-PLUS 3.4 for specific examples, sometimes by orders of magnitude. These claims were usually true. However, most such examples involved very small data sets where the new memory management features of Sv4 have no effect. Inevitably, there is a cost overhead associated with the new memory management features designed to improve performance with large data sets, which makes a typical function call slower when dealing with small objects. For example, this might make a typical function call take 9 seconds rather than 7 seconds, say, for about a 25% decrease in speed. The designers felt that the penalty of having to wait an extra couple of seconds for small problems such as this was worthwhile for the benefit of significant speedups (in the order of many seconds to minutes and even hours) when working with large objects where the performance improvements can have a dramatic effect.

A number of bugs in the Sv4 engine went undetected before the S-PLUS 5.0 release, which meant that S-PLUS 5.0 was somewhat unstable compared to S-PLUS 3.4, and crashes occurred with lamentable frequency. Many of these problems were resolved for S-PLUS 5.1, and Insightful has continued to investigate and fix such bugs for the S-PLUS 6 release.

A major goal of the S-PLUS 6 release has been to eliminate the performance problems introduced in S-PLUS 5 for UNIX, which in turn means improved performance for both S-PLUS 6 for UNIX and S-PLUS 6 for Windows. The aspects of the Sv4 language which make such improvements possible are discussed in the following sections.

### Memory consumption in S-PLUS 6

In any discussion about the performance of S-PLUS, it's necessary to consider the issue of memory. Memory (or lack of it) is the root cause of almost all performance problems. This is because modern operating systems such as Windows 95/98/2000 and all UNIX-like operating systems including Solaris and Linux make use of **virtual memory**. This system allows your PC or workstation to simultaneously run programs whose total memory requirements exceed the amount of available physical RAM (as provided by the RAM chips on your PC motherboard). The details are sometimes complex, but it basically works by copying portions of data and programs

which haven't been used recently to disk, freeing up physical RAM for other applications which are running at present.

Typically, the operating system will allocate a certain amount of disk (the **swap space**) for this purpose; the total amount of virtual memory is therefore the sum of the swap space and the physical RAM. If the active memory requirements of all applications exceeds the amount of physical RAM, the PC may be spending much of its time transferring blocks of memory between the physical RAM and the swap space. This process is very slow compared to working in physical RAM alone, and the worst-case scenario ("thrashing") can make your PC appear to hang. Therefore, reducing the amount of memory an application requires will generally allow it to run faster, because less swapping between physical RAM and disk will be required.

In the context of S-PLUS, it is important to consider not only the processing time of a certain function or script, but also the amount of memory it requires to run. Using less memory not only means the function will generally run faster, but also that the function will scale better to large data sets.

S-PLUS 6 for Windows uses about 35Mb of virtual memory at start-up (as measured by the optional "VM Size" field of the Windows 2000 Task Manager), compared to about 26Mb for S-PLUS 2000. S-PLUS is a large and complex application, and most users will in general require only a fraction of its capabilities. As a result, when running S-PLUS you will probably find that the physical RAM requirements drop as parts of S-PLUS which you don't use get swapped out to disk. For example, the S-PLUS executable includes a number of large Fortran common blocks associated with some statistical computation routines new to S-PLUS 6. Once swapped out to disk, this memory will never be swapped in again unless that particular routine is used. As another example, when using only the command-line window, most of the S-PLUS 6 GUI is untouched and will get swapped out to virtual memory; it is not unusual to see S-PLUS run with as little as 12MB of physical RAM in this scenario.

For users who *never* use the GUI, with S-PLUS 6 for Windows we now provide a "console" version of S-PLUS. This provides the S command-line only, without the S-PLUS Graphical User Interface. This console version is ideal for using with Emacs Speaks Statistics (ESS)<sup>4</sup> which provides command-line and script editing facilities for S. The console version of S-PLUS requires only 23MB of virtual memory, of which as little as 9MB needs to be in physical RAM in order to run. (Unfortunately, the console version provides no interactive graphics device at present, but a Java-based interactive graphics device is available in the **winjava** library, available for download from the Insightful website<sup>5</sup>.)

Because of the increase in the memory footprint for S-PLUS 6 on Windows compared to S-PLUS 2000, we have increased the recommended minimum RAM from 32MB to 96MB. This reflects not only the increase in virtual memory required by the S-PLUS executable but also the fact that the latest releases of the Windows operating system (in particular, Windows 2000) require much more physical RAM than previous versions of Windows. Although S-PLUS will run fine on a PC with only 64MB of RAM (quite modest by modern standards), the 96MB recommendation is intended to give the S-PLUS user sufficient free physical RAM to be able to perform computations on datasets of a reasonable size.

The Sv4 engine included in S-PLUS 6 allows you to manipulate and analyze much larger datasets than in Sv3. In the sections that follow, we discuss two features new to Sv4 – *memory mapping*

---

<sup>4</sup> ESS can be freely downloaded from <http://www.stat.math.ethz.ch/ESS/>. It requires FSF GNU Emacs.

<sup>5</sup> <http://www.insightful.com/downloads/libraries>

and *reference counting* – which allow for the processing of large data objects without unnecessary accumulation of memory.

## Memory mapping

The Sv4 uses memory mapping for large objects stored in databases. Memory mapping allows the operating system to refer to objects stored as files in the `.Data` directory (or any other database) by seeking into the file directly, instead of copying the entire contents of the file to virtual memory. It is particularly useful when you want to read a small portion of a very large object. Because the operating system limits the number of memory mapped files which may be used at once, S-PLUS disables memory mapping by default. You can enable it and set a limit for the minimum size of object that will be memory mapped with the function `mmap.control`; for example, `mmap.control(50000)` sets enables memory mapping for objects 50KB and larger<sup>6</sup>.

As an example with memory mapping enabled, suppose that the object `x` is a double vector of one million elements (requiring 8MB of storage), created say with the statement

```
> X <- rep(1.1, 1000000)
```

The first time `x` is referred to in a new S-PLUS session, for example with

```
> temp <- X[800000:801000]
```

a memory map is created for the object `x`, which allows references to the object such as the one above to refer directly to the database file on the hard drive instead of loading the contents of `x` into virtual memory. Because `x` is memory mapped, no extra virtual memory is required by the operating system to refer to `x`. What the operating system *reports* when memory maps are used depends on the operating system: Windows 2000 will show no increase in the total size of the S-PLUS process nor any decrease in the amount of virtual memory available; Solaris will show an 8MB increase in the size of the S-PLUS process but no decrease in the amount of VM available. In neither case does the total amount of virtual memory available to S-PLUS or any other process decrease.

In addition to using memory maps for large objects created by the user, S-PLUS also uses memory maps to refer to each entire database included in the list of chapters attached at startup (all of the objects in each database are stored as a single large file in the `.Data` directory). This allows S-PLUS to refer to all of the standard functions and datasets without requiring any extra virtual memory for these objects when S-PLUS starts. However as mentioned above, on some operating systems such as Solaris these memory maps are reported as an increase in the size of the S-PLUS process even though the amount of virtual memory available is not reduced. This is one reason why the size of the S-PLUS 6 process on some UNIX systems appears so much larger than the S-PLUS 3.4 process was on the same system.

## Reference counting and memory management

One particular efficiency feature of the Sv4 engine featured in S-PLUS 6 is the notion of reference counting. The basic principle is this: whenever an object is assigned (copied) to another name, it is just a *reference* (or header) that is copied. For example, after the assignment

---

<sup>6</sup> Memory mapping of user-defined objects currently not available on the Windows platform. However, memory mapping is used for system objects automatically.

```
y <- x
```

the objects  $y$  and  $x$  both “point” to the same area of memory; the assignment itself requires no extra memory over and above that already allocated for the contents of object  $x$ , other than a small amount of header information for  $y$ . Internally, this is managed by maintaining a *reference count* for each block of memory; every time a new object points to that block of memory, the reference count is increased, and every time an object pointing to a block of memory is destroyed (because its frame is no longer in scope, or because it was specifically destroyed using the function `unset()`), the reference count is reduced by one. Once the reference count reaches zero, that block of memory is immediately reclaimed by the engine. Reference counting is thus similar to the garbage collection process employed by R<sup>7</sup> version 1.1, except that the process of reclaiming memory happens continuously instead of when the memory has grown to a pre-specified size.

What happens when modifying an object whose reference points to a block of memory which is referenced by other objects? Sv4 uses the *copy-on-write* principle: when an object is modified, where necessary a copy is made to distinguish it from other references and the reference count of the original copy is reduced by one. So, if  $x$  and  $y$  in the above examples are vectors, then the replacement statement

```
y[1] <- 1
```

causes the memory area referred to  $y$  to be copied (this increasing the memory requirements of S-PLUS by the size of  $y$ ) and then modified. The value of  $x$  is unaffected.

Referencing also occurs when including one object inside another list, as for example, in

```
z <- list(e1=y)
```

or in function calls, such as in

```
f(y)
```

In both cases, the contents of  $y$  are not copied until the `e1` component of  $z$  (in the first example), or the value of argument 1 in the frame of  $f$  (in the second example) are modified. In other words, arguments to functions are now implicitly passed using a call-by-reference mechanism, rather than call-by-value. From the S programmer’s perspective, all of this is completely transparent: you can still assume from a functional perspective that all assignments create copies, and that all function arguments are call-by-value and not call-by-reference. The reference counting mechanism operates automatically to keep memory usage to a minimum. The one exception where user intervention is beneficial is the use of the function `unset()`, which removes an object and any references to any memory it occupied, reclaiming that memory if it is not referenced by any other object.

## Reference counting example

Consider the following function  $f$ , which we will call like this

---

<sup>7</sup> An independent open source implementation of a variant of the S language. More information on R is available from <http://www.r-project.org/>

```
X <- rep(1.0, 1e6)
f(X)
```

The argument to `f` is a double-precision vector of one million elements, requiring 8MB of storage. The definition of `f` is as follows:

```
f <- function(x) {
  print(x[1:10])
```

*The formal argument `x` is a reference to the database object `x`, which has already been memory-mapped. Calling `f(x)` does not increase the memory requirements of S-PLUS, since `x` is not copied for the call to `f`.*

```
x[1:10] <- rnorm(10)
```

*`x` is modified here, and so a copy must be made at this stage, for a memory increase of 8MB.*

```
y <- list(alpha=x, beta=1:10)
```

*The object `y` created here includes a reference to memory shared by `x`; the memory referenced by `x` is not copied again at this stage. There is a negligible increase in memory for the list structure of `y` and its second component. Total memory: 8MB.*

```
z <- list(gamma=y)
```

*`z` includes a reference to the list structure included in `y` for zero memory increase. Total memory: 8MB*

```
z$gamma$alpha[1] <- 123.4
```

*At this stage, the memory block referred to by both `x` and `z$gamma$alpha` is copied (increasing memory by 8Mb), and that copy is now referenced from the `gamma$alpha` component of `z`. Total memory: 16Mb.*

```
unset(z)
```

*`z` and its components all have their reference count reduced by one, resulting in the freeing of memory associated with `z` and in particular its `gamma$alpha` component. The memory consumption reduces by 8Mb for a total of 8Mb.*

```
return()
```

*After `f()` returns, the memory associated with the argument `x` is no longer required as both `x` and `y` are out of scope (their frame is destroyed), and the memory requirement drops by 8Mb back to zero.*

```
}
```

## Performance metrics for large and small objects

The new Sv4 engine included in S-PLUS 6 radically changes the way computations are performed compared to S-PLUS 2000. The main changes that impact on performance are:

- Fewer copies of data objects are typically made during a function call in S-PLUS 6 compared to S-PLUS 2000. This is due to reference counting, memory mapping, and use of the `copy` argument to calls to C and FORTRAN code. This generally reduces memory requirements in S-PLUS 6.
- In S-PLUS 6, additional time is required to set up function calls due to the extra information associated with each object, and the fact that most low-level functions in Sv4 are generic, requiring a more complicated method of invocation than in S-PLUS 2000. This generally means that S scripts which call a large number of function but which operate on small objects will generally run slower in S-PLUS 6 than S-PLUS 2000. However, since the total time required is small the percentage increase could be large. For problems involving larger objects, the benefits of the new system become apparent.

Almost everything S-PLUS does is a function call, and S-PLUS contains more than 4,200 functions. Generalizations about a system as complex as S-PLUS are risky, but the following rules are useful:

- Functions applied to large data objects are generally faster and use less memory in S-PLUS 6 than in S-PLUS 2000. As an example, our benchmarks indicate that `lm()` can handle data sets 2-3 times larger than in S-PLUS 2000, and is faster for data sets larger than about 5,000-10,000 observations.
- Functions applied to small data objects are generally slower in S-PLUS 6 than in S-PLUS 2000. This is due to the overhead of the new class system and reference counting system.
- Loops (especially `for` loops) involving operations on small objects may run slower in S-PLUS 6 than S-PLUS 2000. Loops involving larger objects will generally run with less memory consumption and greater speed than in S-PLUS 6 than in S-PLUS 2000, as they benefit from the effects of reference counting.

## Programming tips for efficient computation

Efficient computation generally means using as little memory as possible for the calculations. Reference counting helps greatly in reducing the number of copies of objects in working memory, but in order to maximize performance these programming tips can help.

**Use vectorized computations.** Looping through the elements of a vector is almost never necessary. Use S's vector arithmetic features to compute on each element of the vector in a single operation. As a trivial example, to add 1 to each element of a vector `x` you should use

```
x <- x + 1
```

and never

```
for(i in 1:length(x)) x[i] <- x[i]+1
```

The one area where vector arithmetic may not suffice is where the computation for each element depends on other elements in the vector. That said, a number of built-in functions can help here, notably `pmax`, `cumsum`, `cumprod` and `cummax`.

**Remove names** where possible. S allows you to associate names with vectors (creating objects of class "named"). Deleting the names (and perhaps restoring them later) can have a marked increase on performance in computationally intensive loops.

**Pre-allocate memory** if you can. It is usually better to allocate an entire matrix and modify the elements of that matrix, than to build up a matrix row by row using `rbind` (or equivalently, by column using `cbind`).

**If you need to accumulate data in a loop, use a list.** Because each element of a list is individually reference counted, this will result in the least build-up of a memory. If necessary, convert the list to another data type (such as matrix or data frame) at the end of the loop.

**Use the "apply" family of functions for looping** wherever possible. The functions `apply`, `lapply`, and so on, have been optimized to make best use of S-PLUS's memory reclamation strategies.

**When using a for loop, encapsulate the body of a loop in a single function call.** S-PLUS reclaims the memory allocated inside a function (that would not otherwise be reclaimed through reference counting) when the function call ends. By encapsulating the body of the loop in a function call you ensure the maximum amount of memory is reclaimed at each iteration.

More programming tips such as these may be found in the S-PLUS 6 Programmer's Guide.

## 6. CONNECT/C++ and the Windows GUI

S-PLUS 6 for Windows is essentially two applications in one: the Sv4 "engine" which provides all of the data objects and language capabilities, and the S-PLUS Graphical User Interface (GUI), which provides the menus and dialogs, command, script, report and graph windows, and the Object Explorer.

In S-PLUS 2000 these two applications operated somewhat independently. For example, displaying a data set in a data window required the engine to provide a copy of the data set to the GUI, which would then display it. Modifications to the data set in the data window would then have to be "notified" to the engine in order to update the data. This meant that two copies of the data were required (one for the engine and one for the GUI), and there was a potential for a mismatch between the engine and GUI data sets. As another example, the Object Explorer in S-PLUS 2000 could be very slow when filtering on databases with a large number of large objects, as it had to bring each of these objects into memory in turn in order to determine their attributes.

In S-PLUS 6, the engine and the GUI are much more tightly integrated. This is because the GUI is programmed using the new CONNECT/C++ feature of S-PLUS 6. This allows the GUI to access data and objects directly in the Sv4 engine, without requiring an extra copy of the data. As a result, communication between the GUI and the engine require much less in the way of additional memory, and is much faster. In turn, the GUI in S-PLUS 6 is much more responsive than it was in S-PLUS 2000: for example, the Object Explorer can now deal efficiently with hundreds of large objects.

The CONNECT/ C++ interface which made this possible is also available to C++ programmers using S-PLUS 6. Using CONNECT/ C++ you can build your own C++ application which makes use of S objects, expressions and function calls. In addition, we provide Microsoft Visual C++ wizards which make creating such applications easier, and also allows for live debugging of C++ applications linked to S-PLUS.

## 7. Conclusion

S-PLUS 6 represents a significant milestone in the evolution of S-PLUS, building on its reputation as the most flexible, interactive environment for analyzing, visualizing and presenting data. Offering cross-platform compatibility, numerous new features and improved performance, S-PLUS 6 delivers great benefits for all S-PLUS users. For the latest information on S-PLUS please visit our Web site at [www.insightful.com](http://www.insightful.com). As always, we welcome your feedback about S-PLUS and look forward to working with you as we develop new versions.

## Appendix: Recommended reading

For more information about the Sv4 language as incorporated in S-PLUS 5 (for UNIX) and S-PLUS 6 (for UNIX and Windows) we recommend the following:

The S-PLUS 6 Programmer's Guide, included with S-PLUS 6, has information about the new features of the Sv4 language, and changes from Sv3. It also includes a chapter on optimizing performance of S-PLUS functions.

W.N. Venables & B.D. Ripley, "S Programming" (Springer, 2000) is an excellent reference for the S language and its variants. In particular, Chapter 5 gives an overview of the object-oriented capabilities of the new Sv4 language featured in S-PLUS 6. At a more practically-oriented level, Venables & Ripley, "Modern Applied Statistics with S-PLUS" (3<sup>rd</sup> Ed, Springer, 1999) gives a comprehensive overview of the S language as a whole, plus provides extensive documentation and examples of the statistical features of S-PLUS.

J.M. Chambers, "Programming with Data" (Springer, 1998) gives a detailed account of the new features of the Sv4 language, although there are a few inconsistencies relative to S-PLUS 6 resulting from changes made to S for S-PLUS. Generalities of the S language which are common to Sv3 and Sv4 are not covered here, so Becker, Chambers & Wilks "The New S Language" (Chapman and Hall, 1988) makes a useful companion to this book.

A list of further titles relating to S-PLUS is available at <http://www.insightful.com/support/splusbooks.asp>.

For information on site visits or purchasing S-PLUS contact [sales@insightful.com](mailto:sales@insightful.com) or call 206-283-8802 or 800-569-0123.