A Developer’s Guide to Protecting Memory
Detect and Eliminate Damaging Memory Leaks with ABAP Memory Inspector

Memory is a critical resource, but some developers still underestimate its value. The cost of memory dominates the price of today’s servers — in general, approximately 80% of hardware costs are attributed to memory. Yet even when developers work to use memory efficiently, the problem of undetected memory leaks often remains.

Memory leaks occur during long-lasting stateful sessions (when a series of business objects is processed consecutively); memory is occupied, but not freed, once it is no longer needed. This all-too-common problem results in high memory consumption, tying up memory so that other users or programs can’t use it.

In my previous Performance & Data Management Corner column, I described a few options for solving memory leaks on software architecture and programming levels.¹ Now I’d like to familiarize you with a tool that will allow you to analyze those memory leaks — and ultimately eliminate them. This tool is the ABAP Memory Inspector.²

Before we dive into ABAP Memory Inspector’s functionality and how to use it, let’s investigate some familiar SAP tools that developers can employ to find and identify user sessions with high memory usage, the main hiding places for memory leaks.

Identify Potential Memory Problems
For developers, the first step to finding and stopping memory leaks is to take advantage of some of the transactions that SAP NetWeaver offers:

- **SM04** (User session memory list): To look for memory leaks, you need to monitor memory consumption while several consecutive business objects are processed within the same user session. Transaction SM04 shows the current memory usage of a user session. SM04 is ideal for online, real-time monitoring. For example, you can easily monitor online the memory growth of a long-running dialog transaction or batch job (see sidebar for more information about using SM04).

- **STAD** (SAP Workload: Business Transaction Analysis): STAD is ideal for monitoring the resource consumption of a transaction after processing has completed. To find the maximum level of memory consumption of a dialog transaction or a batch job after it has finished, check the statistical records — which are recorded by default — in the SAP Workload Monitor using the STAD transaction. Once you enter filter criteria (your user name, for instance), you’ll see a list of the dialog steps you processed. This list contains a “Memory Used (kB)” column, which shows the value of the memory consumed at the point in time when the dialog step finished processing.

Use the ABAP Memory Inspector to Analyze the Problem
Once you have used these tools to pinpoint the source of your memory problem, you can further analyze the problem with ABAP Memory Inspector by taking snapshots of the memory usage of a user session.³

To find memory leaks in the ABAP virtual machine (VM), take two memory snapshots at different times — for example, one after the nᵗʰ purchase order has

[misc]


² For a detailed introduction to ABAP Memory Inspector, see “Analyze Memory-Related Problems in Your ABAP Programs in Less Time and with Less Effort Using the ABAP Memory Inspector” in the November/December 2004 issue of SAP Professional Journal (www.SAPpro.com).

³ To be more precise, this snapshot is of a single internal session, which is what a user session typically consists of. Though the ABAP language keywords CALL TRANSACTION and SUBMIT REPORT create a stack of internal sessions, ABAP Memory Inspector only captures one internal session of this stack per snapshot.
Working with the User Session Memory List (Transaction SM04)

While the initial SM04 session list already contains memory consumption information, it does not show all relevant details. For example, the granularity of the display is only 1MB. To combat this, you can change the display to byte level by choosing Goto → Memory. In this detailed list, however, multiple user sessions might still be added together in one entry. This happens when multiple user sessions were created through one SAP GUI logon. This causes a problem if you’re looking to drill down and monitor only one session (see figure below). To obtain memory information for a selected user and one particular session only, mark that user and choose Goto → Memory List → Session Details.

Please note: In these detailed lists, overall memory is broken down into four internal categories: Roll, Page, Mem(Total), and Mem(Priv.). To obtain the overall memory usage, add these values together.

been processed and another after the \((n+1)^{\text{th}}\) purchase order has been processed in one session. ABAP Memory Inspector will then allow developers to analyze facets of these snapshots, calculating the difference in memory allocation between them and displaying only those memory objects that are new or have changed in size. This way, you’ll be able to pare down the number of memory objects that you’ll need to analyze further.

ABAP Memory Inspector also only lists dynamic memory objects — in-memory structures that are created or expanded in size during run time. Only those types of memory objects can cause memory leaks. They include the following:

- Bodies of internal tables (as created by statements like DATA itab TYPE TABLE OF <struc>)
- Strings (as created by statements like DATA lstring TYPE STRING)
- ABAP objects or class objects (such as a class created by the CREATE OBJECT statement)
- Anonymous data objects (such as a data type created by the CREATE DATA statement)

Methods of Taking Memory Snapshots

There are several ways to trigger a memory snapshot. For SAP GUI transactions, the easiest way is to use the System menu available in all SAP GUI-based screens through the path System → Utilities → Memory Analysis → Create Memory Snapshot.

To take a memory snapshot of a non-SAP GUI application or to take the snapshot when the maximum memory consumption is reached (during the processing of a screen, for example), you will have to use the ABAP Debugger.4

What About Taking Memory Snapshots in Java Environments?

The main difference between ABAP memory snapshots and those taken in the Java VM (called heap dumps in the Java world) is that, in the ABAP VM, the content of one particular user session can be analyzed separately, whereas the Java heap dump contains the memory objects of the whole VM.

An equivalent tool to ABAP Memory Inspector for a Java environment, SAP Memory Analyzer, is available. For more information, please visit www.sdn.sap.com/irj/sdn/wiki?path=/display/Java/Java+Memory+Analysis.

4 For another method of triggering memory snapshots, see the online version of this article at www.SAPinsideronline.com.
With the Classic ABAP Debugger, create a snapshot through the menu path Development → Memory Analysis → Create Memory Snapshot.

In the New ABAP Debugger, the memory analysis functionality is not part of the default desktop and must be explicitly loaded into one of the user-specific desktops via the Replace Tool functionality (selecting Special Tools → Memory Analysis in the list of available tools). Once this is done, you can take the snapshot as shown in Figure 1 by calling the Create Memory Snapshot service of the tool.

Methods of Analyzing Memory Snapshots
The simplest way to analyze memory snapshots — no matter how you created them — is to call the ABAP Memory Inspector with transaction S_MEMORY_INSPECTOR via the command field or to use the menu path System → Utilities → Memory Analysis → Compare Memory Snapshots.

However, if you start the ABAP Memory Inspector directly from the Classic ABAP Debugger (via Development → Memory Analysis → Compare Memory Snapshots), you will gain a special advantage: The ABAP Memory Inspector session and the ABAP Debugger session are linked, making it possible to display the content or the value of the memory object found in the ABAP Memory Inspector directly in the ABAP Debugger. To do this, right-click on the object in the ABAP Memory Inspector session and select Display in Debugger, then switch to the ABAP Debugger screen and press ENTER.

With the download and upload functionality of ABAP Memory Inspector, you can also analyze snapshots in any other system that is running on the same SAP NetWeaver release as the system under test.

Further Investigating Memory Snapshots
ABAP Memory Inspector enables you to select up to two memory snapshots — snapshot t_0 and snapshot t_1. You can then choose to analyze the memory usage in each snapshot individually by selecting (t_0) or (t_1) in the dropdown list in Figure 2, or the memory change between the two (t_1 - t_0). The system automatically selects the snapshot with the later timestamp as snapshot t_1. This means that each positive value in the comparison t_1 - t_0 indicates an increase in memory consumption.

For each snapshot or comparison of two snapshots, you can display different views, such as the Overview and Ranked List by Type views, by clicking on the View dropdown list.

Let’s break down the values displayed in Figure 2:
- **MM_TOTAL** shows the total size of one internal session at the time the snapshot was taken. Compare this with the values retrieved by SM04 or STAD to verify if you are using the correct internal session.
- **ABAP_TOTAL** displays the amount of memory that can be directly assigned to a single memory object like an internal table. The difference between MM_TOTAL and ABAP_TOTAL indicates the administrative part of the ABAP runtime system or shows that some runtime structures, like directories for memory objects, cannot be assigned to a single ABAP memory object. But reducing the total number of ABAP memory objects will reduce the quantity of the ABAP_TOTAL and the MM_TOTAL values, since the corresponding memory required for administrative data is also reduced.
- **Delta MM_Total (GC)** shows the amount of memory that is freed during the ABAP Memory Inspector’s...

---

**Note!**
When working in the New ABAP Debugger window, do not use the menu path System → Utilities → Memory Analysis → Create Memory Snapshot. Memory snapshots created this way only contain memory objects of the ABAP Debugger session rather than the user session you are trying to analyze.
automatic garbage collection (GC). The Memory Inspector triggers this GC before a snapshot is taken to avoid including dead objects in the snapshot. If this value makes up a significant part of the overall user session (MM_TOTAL), investigate which classes these ABAP objects belong to and try to uncover why they produce such a large amount of garbage. You can then take steps to optimize these programs — by reusing objects, for example.

**Compare Memory Snapshots**

After developers have collected enough information about memory allocations and relevant memory values, they can interpret and analyze the snapshots by comparing them in ABAP Memory Inspector.

There are some important aspects to keep in mind when taking snapshots:

- Take both snapshots at defined, identical points in your business process — for example, one before and one after a business object was processed (when processing open purchase orders from a work list, for instance).

- Do not take snapshots for the first business object or work package being processed. Instead, you should start at a time when all programs have already been loaded.

- Make sure exactly the same business process is executed for each snapshot and that no additional functionality is run. Otherwise it will be hard to distinguish whether a new memory object in the snapshot is due to a memory leak or the result of additional business functionality. For example, do

---

**Key Terms**

- **Bound size** is the quantity of memory that will be freed — immediately or after a garbage collection (GC) run — by deleting a particular memory object or all references to it.

- **Referenced size** is the sum of the size of all memory objects that can be reached by following all references from a memory object to all related memory objects.

---

\(^5\) See Key Terms box or the online version of this article for more details.
not take the first snapshot for a purchase order with standard trading goods and the second snapshot for a purchase order with configurable materials.

Remember that if you start ABAP Memory Inspector directly from the Classic ABAP Debugger, you can display the content of memory objects directly within the Debugger. Load the two relevant memory snapshots and switch to the Ranked List by Type view. In this view, memory objects of the same type — such as all ABAP objects that belong to the same class — are grouped together by default, resulting in a manageable list with fewer entries to be analyzed. However, you can expand the list by manually drilling down into individual ABAP objects.

Once you reach the screen shown in Figure 3, there are two approaches you can take to compare your memory snapshots.

**Approach #1 — Identify the Largest Individual Objects Contributing to a Memory Leak**

To find the largest individual contributors to a memory leak, sort the ranked list in the memory snapshot according to the values in the Bound (Allocated) column in descending order (see the Key Terms box on the previous page to learn more about what these column titles mean). This divides the table into three categories: red, black, and blue (refer again to Figure 3).

- The top of the red list shows the largest new memory objects. Developers should examine each object closely to figure out if it is needed or if it can be deleted. By expanding the memory object in the ABAP Memory Inspector, the full information about all references to the memory object is displayed. More than one reference often exists, especially for ABAP objects, because developers sometimes forget to clear unneeded references, thus keeping the memory object alive. If developers need to follow the hierarchy of these references upward beyond the single level shown in the memory snapshot to locate the root cause of a memory leak, they must use the ABAP Debugger.

- The top of the black list shows the memory objects with the largest growth. Developers should view the content of each of these memory objects in ABAP Debugger to see if, for example, data belonging to the previously processed business object is being kept unnecessarily.

- The blue list contains the objects that were no longer alive when the t_1 snapshot was taken — indicating that they were no longer taking up memory. Here, developers should look for corresponding entries for these memory objects in the red list because that would indicate that one instance of the memory object was deleted and a new, identical instance (with a new ID) was created. If the size of both memory objects is identical at times t_0 and t_1, then these objects are not causing a memory leak. Nevertheless, in order to save CPU time, developers should consider reusing the memory object instead of deleting and re-creating it.
Approach #2 — Identify the Largest Object Cycles Contributing to a Memory Leak

An alternative, top-down approach is to look at your memory snapshot with a focus on finding the largest object cycles. To do so, sort the list in descending order according to the Referenced (Allocated) column. The aim of this approach is to find larger networks of memory objects that are kept alive, but may no longer be needed.

Often, in some leading object, the decisive reference is kept alive needlessly. By releasing this reference, a whole network of objects can be deleted at once. This is the advantage of a top-down analysis of the largest object cycles. On the other hand, not every memory object that is inspected may be a good candidate to throw away since that object may still be alive on purpose.

Outlook and Conclusion

Memory is a valuable resource and should be treated as such. Since memory leaks do not easily show up in single user tests — which are often done during development — it’s up to developers to proactively look for memory leaks and ensure that they aren’t slowly draining away a crucial resource.

SAP offers a variety of tools — from basic transactions, such as SM04 and STAD, to the ABAP Memory Inspector — to not only detect memory leaks, but to analyze them and eliminate their sources as well.

SAP is continuing to provide developers with enhanced functionality for memory conservation. With SAP NetWeaver 7.0, for example, developers will be able to take memory snapshots on ABAP Shared Objects areas, like application-managed buffers, which are becoming more and more popular — and therefore are a potential source of memory leaks.

With existing tools and continued SAP support, developers can rest assured that they are well armed to battle memory problems now and in the future.

Additional Resources...

...from SAPinsider


