



---

SIEBEL SYSTEMS, INC.

*Siebel Application  
Response Monitoring*

**SIEBEL**  
Sales • Marketing • Customer Service

---

# Siebel Application Response Measurement 7.5.3 Technical Note

<b>Summary .....</b>	<b>3</b>
<b>Overview of Siebel Application Response Monitoring.....</b>	<b>4</b>
Architecture of Siebel Application Response Monitoring 7.5.3.....	5
Overview.....	5
SARM areas of instrumentation .....	5
Request execution path – Example .....	6
<b>Enabling SARM .....</b>	<b>9</b>
Enabling SARM on the Web Server .....	9
Enabling SARM on the Siebel Server.....	11
<b>SARM Analyzer Tool.....</b>	<b>13</b>
SARM Analyzer Syntax.....	13
Performance Area Aggregation Analysis.....	15
Example output .....	18
Call Map Generation .....	24
Example output .....	25
User Session trace.....	26
Example output .....	28
SARM Binary File to CSV Conversion.....	32
<b>Best Practices .....</b>	<b>35</b>

# Siebel Application Response Measurement 7.5.3 Technical Note

## Summary

This Technical Note discusses the architecture and usage of Siebel Application Response Measurement (SARM) a service that has been introduced as part of Siebel 7.5.3. SARM is modeled after ARM, the industry standard for application response time measurement.

SARM can be used in both pre- and post production systems to identify or troubleshoot response time problems.

This Technical Note provides basic examples of SARM usage and describes the currently supported methods of utilizing SARM.

# Siebel Application Response Measurement 7.5.3 Technical Note

## Overview of Siebel Application Response Monitoring

Siebel Application Response Management (SARM) 7.5.3 is a framework for identifying performance problems in the Siebel 7.5.3 solution. SARM allows administrators to collect critical performance and timing data, thereby making it possible to profile the execution of requests throughout the Siebel Server and its various components.

The SARM Analyzer is a post-processing tool that assists Administrators in understanding the performance data captured by SARM. The tool converts the SARM files from a binary representation to a human readable format. SARM Analyzer performs various levels of computation and analysis. Depending on the user options, the tool then generates an XML or CSV output of the call stack and analysis results.

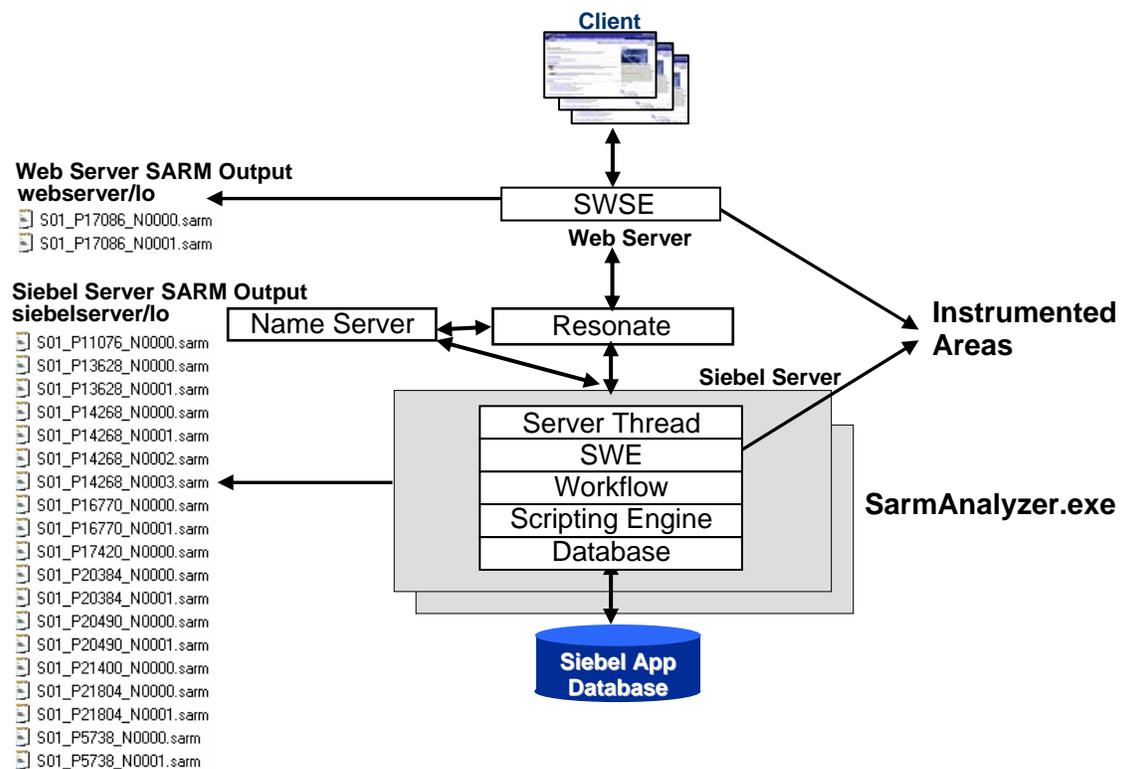


Figure 1. SARM Architecture and areas of instrumentation

# Siebel Application Response Measurement 7.5.3 Technical Note

## Architecture of Siebel Application Response Monitoring 7.5.3

### Overview

Prior to being able to leverage SARM, it is important to understand the areas and sub-areas of the Siebel architecture that have been instrumented to collect response time information. Understanding the type of data and the area of the product where timing information is captured is an essential first step in being able to use SARM and analyze its output.

### SARM areas of instrumentation

Table 1 shows the areas and sub-areas of performance monitoring for SARM within the Siebel Architecture:

Area Name	Area Number	Sub-Area Number			
		1	2	3	4
SarmIO	1				
SWSE	2	Login	SWE Request	Session Manager	
Server Thread (SMI)	3	Request Handling			
DB Connector	4	Execute Query	Write Record	Fetch Next Record	Prepare Statement
Scripting Engine	5	VB Script Execute	VB Script Compilation	eScript Execution	eScript Compilation
Workflow	7	Invoke Method	Process Init	Process Resume	Step Execution
SWE	8	Process SWE Command	Build View		

**Table 1**

#### Area: SarmIO

SarmIO measures the time it takes to write the SARM data from memory to disk.

#### Area: Siebel Web Server Extension (SWSE)

SWSE measures the time duration between entry to the SWSE and messages being sent to the Siebel server. Time spent in the SWSE includes the Siebel Gateway and Resonate time.

##### Sub-area: Login

Time spent to request a user login

##### Sub-area: SWE Request

Time for SWSE to handle a request

##### Sub-area: Session Manager

Time for the Session Manager to handle a request.

#### Area: Server Thread (SMI)

Server thread is the area in the Siebel architecture that handles all Siebel server requests. This is the entry point of a request from the web server to the Siebel server. The time indicates the duration it takes the Siebel server to handle a request.

##### Sub-area: Request Handling

Time to handle a request on the Siebel server side

# Siebel Application Response Measurement 7.5.3

## Technical Note

### Area: Database Connector

Database Connector measures the total time it takes a given request when calling the Database Connector layer.

#### Sub-area: Execute Query

Time to execute a "select" statement

#### Sub-area: Write Record

Time to execute a "delete", "update" or "insert" statement

#### Sub-area: Fetch Next Record

Time to fetch a record from a query

#### Sub-area: Prepare Statement

Time to prepare a SQL statement

### Area: Scripting Engine

Total time it takes to execute a script.

#### Sub-area: VB Script Execute

Time to execute a VB script

#### Sub-area: VB Script Compilation

Time to compile a VB script

#### Sub-area: eScript Execution

Time to execute an eScript script

#### Sub-area: eScript Compilation

Time to compile an eScript script

### Area: Workflow

Total time it takes to execute a Workflow process.

#### Sub-area: Invoke Method

Time it takes to invoke a method

#### Sub-area: Process Init

Time it takes to initialize a process (workflow)

#### Sub-area: Process Resume

Time it takes to resume a process (workflow)

#### Sub-area: Step Execution

Time it takes to execute a step within a process (workflow)

### Area: SWE

The Siebel Web Engine (SWE) executes within the context of the Siebel Object Manager.

Therefore any time spent in the SWE is a subset of time of the total Siebel Object Manager time.

#### Sub-area: Process SWE Command

Time it takes to process a request submitted to SWE

#### Sub-area: Build View

The SWE assembles the Siebel View (web page). The Object Manager then sends it to the Siebel Web Server Extension running on the Web Server so it can pass the web page onto the client. This metric reflects the time it takes to assemble/build the view.

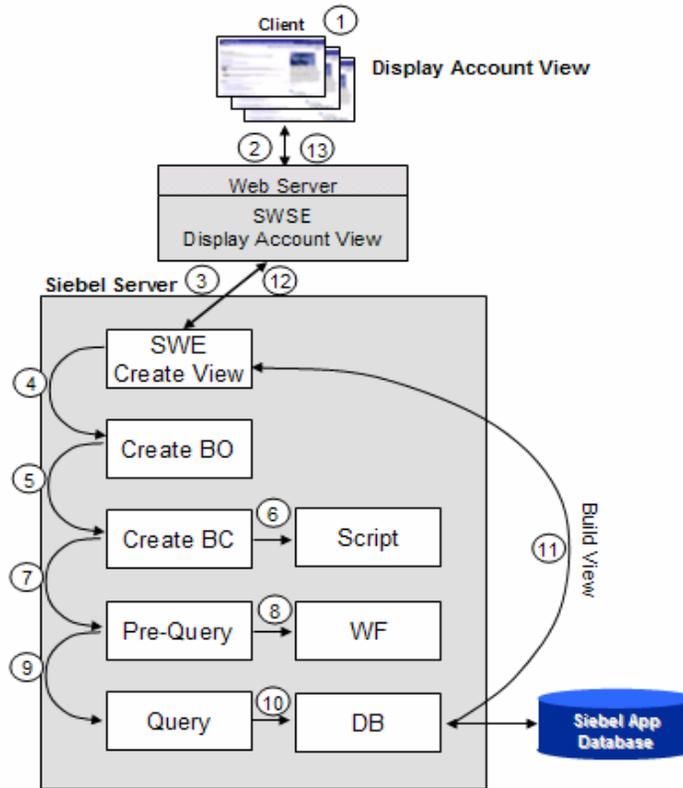
### Request execution path – Example

Below is an example of the execution flow of a request when a user navigates to the Accounts view.

1. User clicks on the Accounts view.
2. A request is submitted from the browser to the Web Server.
3. The Siebel Web Server Extension (SWSE) submits the request to the Siebel Server.
4. Within the Siebel Server, the Siebel Web Engine (SWE) invokes the Siebel Object Manger (OM) to create a Business Object (BO).

## Siebel Application Response Measurement 7.5.3 Technical Note

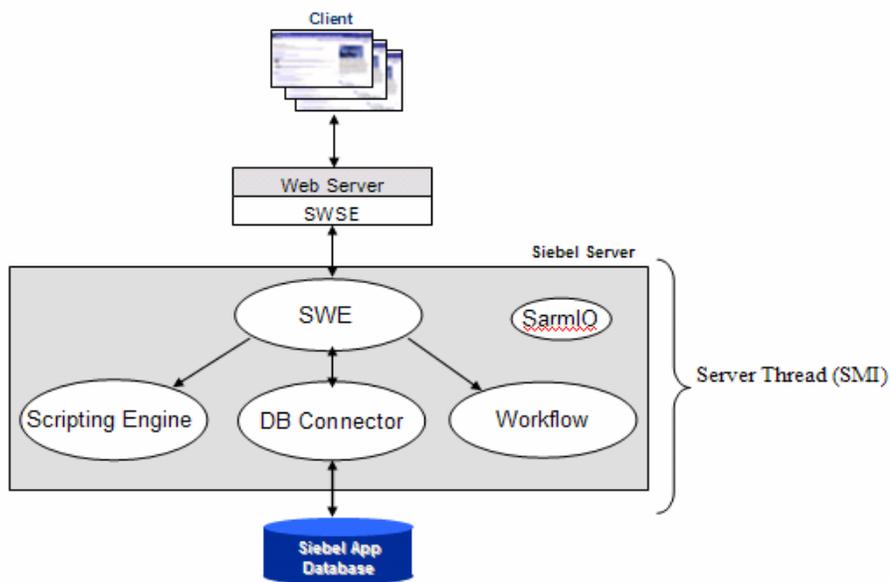
5. The BO creates a Business Component (BC).
6. The BC may invoke a script.
7. The BC may invoke a pre-query function
8. The pre-query function may invoke a workflow.
9. The pre-query function invokes a query function.
10. The query function invokes the Siebel Database.
11. The Siebel database retrieves the account information and sends the data back to SWE.
12. SWE builds the account view and sends the information to SWSE.
13. SWSE sends the account view information to be rendered in the client's browser.



**Figure 2 – Requests Process Flow**

The path of the SARM instrumentation as a result of the request to display the Account's view will be the following:

## Siebel Application Response Measurement 7.5.3 Technical Note



**Figure 3 – SARM instrumentation for an Account's view request**

1. The first response time instrumentation point once the request is submitted from the browser to the web server is SWSE.
2. SWSE submits the request to SWE. SWE will be the second instrumentation point for the request.
3. The request invokes a script, a workflow process and it accesses the Siebel database to retrieve the account information. Based on the path of the request, SARM captures response time information for the Scripting Engine, DB connector and Workflow process.
4. The total response time for the Siebel Server to handle the request is being captured by the Server Thread.

# Siebel Application Response Measurement 7.5.3 Technical Note

## Enabling SARM

SARM is controlled through Siebel server parameters and environment variables. To enable SARM, three parameters need to be enabled for both the web server via environment variables and the Siebel Server via the Server Manager. When a component starts up in the Siebel Enterprise, it will check for the status of the SARM parameters.

There are 3 parameters that control the behavior of SARM:

- **SARMEEnabled:** indicates whether SARM is enabled or disabled for a Siebel Server Component. It is a boolean value (true | false). The default value is 'false'. This parameter can be set at the Siebel Server or Siebel Server Component level.
- **SARMMaxMemory:** SARM 7.5.3 uses a shared memory segment to store the data collected from the Siebel Server Components. Once the in-memory data size reaches a threshold defined by the parameter SARMMaxMemory, SARM will write the data to a file on the local disk subsystem. The default value is '500000', about 5 MB and is specified in bytes.
- **SARMMaxFileSize:** Specifies how large a file gets before SARM will start a new file. SARM will continue to append file segments to the current file until the specified size is reached. When the file limit is reached, SARM will start a new file. The default value is '20000000' (about 20 MB) and is specified in bytes.

SARM is disabled by default. To enable SARM, set the **SARMEEnabled** parameter to true. This can be done on the Web Server, the Siebel Server, or both. When enabling SARM it is important to also consider the appropriate settings for the **SARMMaxMemory** and **SARMMaxFileSize** parameters since these will determine how soon SARM flushes its data to disk, and how large the SARM files will be. Recommendations for setting these parameters are discussed later in this section.

### Enabling SARM on the Web Server

Enabling SARM on the Web Server is done by setting environment variables.

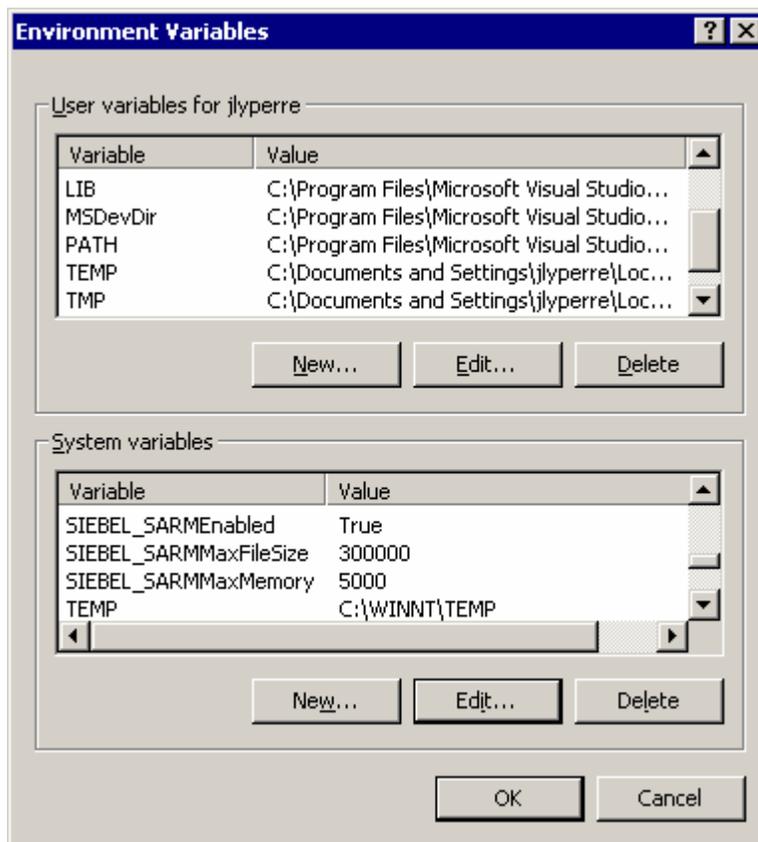
An example on Windows is:

1. From the Desktop, right click on My Computer icon and select Properties.
2. Go to the "Advanced" tab.
3. Select "Environment Variables." Add the system environment variables.

```
SIEBEL_SARMEEnabled      = true
SIEBEL_SARMMaxMemory     = 20000
SIEBEL_SARMMaxFileSize   = 400000
```

4. Note that the equal signs indicate the values that the system variables should be set to. The optimal values vary by specific deployment and may be different than depicted here. On Windows, the machine needs to be restarted so that the settings can take effect.

## Siebel Application Response Measurement 7.5.3 Technical Note



**Figure 4 – Setting environment variables for SARM**

**Note:** Figure 4 shows a sample value for SARM environment variables. The default values for SIEBEL\_SARMMMaxFileSize and SIEBEL\_SARMMMaxMemory are 3000000 (about 30 MB) and 5000 (about 5 KB), respectively.

UNIX Example:

1. Enter the corresponding shell commands to enable the environment variables. For example if using C-shell, use the following command:

## Siebel Application Response Measurement 7.5.3 Technical Note

```
setenv SIEBEL_SARMEEnabled true
setenv SIEBEL_SARMMMaxMemory 20000
setenv SIEBEL_SARMMMaxFileSize 400000
```

Once the values have been modified the web server needs to be re-started for the changes to take effect. To disable SARM once it has been enabled on the web server, simply set the SIEBEL\_SARMEEnabled variable to false and restart the web server process/service.

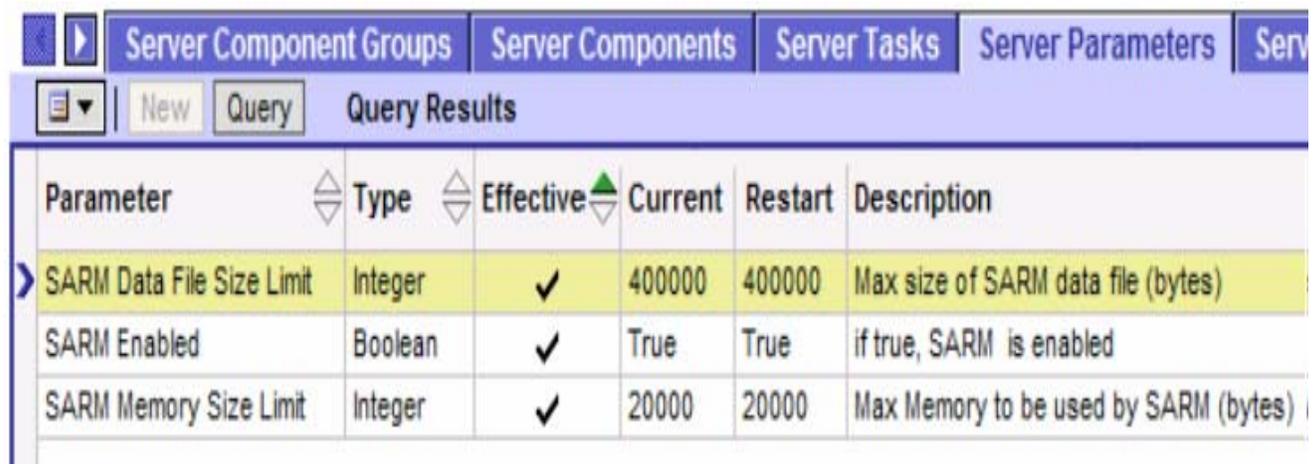
**Note:** If the web server and the Siebel Server are running on the same machine, the maximum memory and files size on the web server will override the value of the Siebel Server. However, in most testing and production environments, the web server and Siebel Servers are running on different machines, so the likelihood of this scenario occurring is minimal.

### Enabling SARM on the Siebel Server

Enabling SARM on the Application Server is done by setting Siebel Server parameters via the Server Manager, either through the user interface or the command line interface.

To enable SARM on the Siebel Server using the Siebel Server Manager Graphical User Interface:

1. Go to Site Map → Server Administration → Servers → Server Parameters
2. In the Server Parameters List Applet, query for "SARM\*".
3. Update the values of SARM Data File Size Limit, SARM Enabled, SARM Memory Size Limit accordingly
4. Stop and re-start the Siebel server for the new values to take effect.



The screenshot shows the Siebel Server Manager GUI with the 'Server Parameters' tab selected. A query for 'SARM\*' is executed, displaying a table of parameters. The table has columns for Parameter, Type, Effective, Current, Restart, and Description. Three parameters are listed: SARM Data File Size Limit (Integer, 400000), SARM Enabled (Boolean, True), and SARM Memory Size Limit (Integer, 20000).

Parameter	Type	Effective	Current	Restart	Description
SARM Data File Size Limit	Integer	✓	400000	400000	Max size of SARM data file (bytes)
SARM Enabled	Boolean	✓	True	True	if true, SARM is enabled
SARM Memory Size Limit	Integer	✓	20000	20000	Max Memory to be used by SARM (bytes)

Figure 5 – Setting SARM parameters through Server Manager UI

To enable SARM on the Siebel Server using the command line:

1. Start server manager command-line interface.
2. To enable SARM at the Server level, enter:

## Siebel Application Response Measurement 7.5.3 Technical Note

change parameter **SARMEEnabled** = true for server *Siebel\_server\_name*  
change parameter **SARMMMaxMemory** = 20000 for server *Siebel\_server\_name*  
change parameter **SARMMMaxFileSize** = 400000 for server *Siebel\_server\_name*

3. To enable SARM at the server component level, enter:

change parameter **SARMEEnabled** = true for component *component\_alias\_name*  
change parameter **SARMMMaxMemory** = 20000 for component *component\_alias\_name*  
change parameter **SARMMMaxFileSize** = 400000 for component *component\_alias\_name*

Once the values have been modified the Siebel Server should to be re-started for the changes to take effect. To disable SARM on the Siebel Server once it has been enabled simply set the SARMEEnabled parameter to false and restart the Siebel Server process/service.

**SARMMMaxFileSize** will determine the maximum size of the SARM file but SARM may create a new file when 80% of the file size is reached.

The settings for size of the maximum memory and file size can be adjusted depending on how much SARM data the administrator wants to keep in a file before they are overwritten. This makes it possible to arrange the configuration such that *n* amount of data is maintained in the files once the maximum number of files is created.

In internal Siebel Engineering testing of a Call Center with a 1000 users accessing the application and executing normal operations, the default values of 5MB and 20MB stored between one to two hours of SARM data. For additional data, the sizes should be changed accordingly. The actual amount of data captured and space taken are deployment specific as some deployments may have higher transaction throughput than others.

Once SARM has been enabled, the system will start collecting performance data for each request. Performance data is temporarily stored in memory up to the value of **SARMMMaxMemory** and periodically flushed to disk. The data written to disk is in binary format. After running the SARM Analyzer, the data captured will be in a different precision format depending on the platform used: in a Microsoft Windows environment the data will be captured in microseconds (One millionth ( $10^{-6}$ ) of a second); in a Unix environment the data will be in nanoseconds (One billionth ( $10^{-9}$ ) of a second).

SARM maintains at most four (4) SARM files in disk per component process. In Siebel 7.5.3, this behavior is not configurable. Once the maximum file size is reached (**SARMMMaxFileSize**), the oldest file is removed. If the administrator needs to keep those SARM files for future reference, it is recommended that the files are archived before they are removed.

The SARM file is created when the application encounters the first instrumentation point in one of the instrumented areas (SWSE, Server Thread, Database Connector, Scripting Engine, Workflow, SWE or SarmIO) and SARM has been enabled. Notice that some SARM files have a size of zero. This is because data is written to file when the buffer (**SARMMMaxMemory**) is full and in this case, no data has been written for that process yet.

Note: If no data is written to the disk it means that SARM is enabled but the SARMMMaxMemory level has not been reached. In 7.5.3, the file is only written to when the buffer is full so in the event that the buffer is never filled before the process is stopped, no data will be collected and the file size will remain 0. If this results in no data ever being written to disk, then the SARMMMaxMemory value can be set to a lower value so the buffer is filled and flushed more often.

Examples of the SARM data files are shown below:

# Siebel Application Response Measurement 7.5.3

## Technical Note

S01_P14260_N0000.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P14260_N0001.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P14260_N0002.sarm	-rw-r--r--	qa1	qa	270272	Jul 07 21:54
S01_P15670_N0000.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P15670_N0001.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P15670_N0002.sarm	-rw-r--r--	qa1	qa	270272	Jul 07 21:54
S01_P18790_N0000.sarm	-rw-r--r--	qa1	qa	0	Jul 07 23:05
S01_P19148_N0000.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P19148_N0001.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P19148_N0002.sarm	-rw-r--r--	qa1	qa	270272	Jul 07 21:54
S01_P20382_N0000.sarm	-rw-r--r--	qa1	qa	2624	Jul 07 21:56
S01_P20488_N0000.sarm	-rw-r--r--	qa1	qa	0	Jul 07 21:56
S01_P20862_N0000.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P20862_N0001.sarm	-rw-r--r--	qa1	qa	301760	Jul 07 21:54
S01_P20862_N0002.sarm	-rw-r--r--	qa1	qa	272896	Jul 07 21:55

The SARM files have the following naming conventions:

S01\_P<pid>\_N<filename>.sarm

S01 – constant number

<pid> - Siebel server process id

<filename> - file number. This starts at **0000** and it increments until it reaches 9999, at which point it wraps around to 0000 again. The Most recent **4** files (per process) are retained. An example of this is S01\_P14260\_N0000.sarm.

## SARM Analyzer Tool

SARM Analyzer, the SARM post-processing tool, converts the binary SARM output files to XML or CSV format. The resulting files include aggregated times for further analysis by the Siebel Administrator. The tool **SARMAAnalyzer.exe** is located in the %SIEBEL\_HOME%\BIN directory.

### SARM Analyzer Syntax

The SARM Analyzer performs various types of processing of the SARM binary files. The output syntax will vary according to the processing desired. Please refer to the following sections for detailed descriptions of the functionality and related syntax:

1. Performance Area Aggregation analysis
2. Call Map Generation
3. User-Session Trace
4. SARM Binary File to CSV Conversion

Independent of the mode in which the SARM Analyzer is used, to get help type:

On a Windows environment, enter:  
SARMAAnalyzer -help

On a Unix environment, enter

# Siebel Application Response Measurement 7.5.3 Technical Note

sarmanalyzer -help

# Siebel Application Response Measurement 7.5.3 Technical Note

## Performance Area Aggregation Analysis

Aggregation analysis is performed by executing the SARMAalyzer executable using the `-f` switch. The following sections detail the different usage possibilities as well as details on the output content and format.

### Usage 1

Executing the SARM Analyzer executable aggregates the contents of a SARM binary file by area of instrumentation. The output file will always be `sarm.xml` and it will be placed in the location from where the command is executed.

#### Command Syntax:

On Windows: **SARMAalyzer -f <sarm file>**

On Unix: **sarmalyzer -f <sarm file>**

**Output File Name:** `sarm.xml`

**Example on Windows:** `SARMAalyzer -f S01_P20862_N0002.sarm`

### Usage 2

Executing the SARM Analyzer executable aggregates the contents of a SARM 7.5.3 binary file by area of instrumentation, but in this case by piping the output to a file, the output file will have the name specified and will be stored in the location specified in the command.

#### Command Syntax:

On Windows: **SARMAalyzer -f <sarm file> > <some\_file\_name.xml>**

On Unix: **sarmanalyzer -f <sarm file> > <some\_file\_name.xml>**

**Output File Name:** `some_file_name.xml`

**Example:** `SARMAalyzer -f S01_P20862_N0002.sarm > %HOME%/P20862_N0002_area_agr.xml`

#### Description:

SARM Analyzer provides the capabilities to perform grouping against the performance data captured in SARM files. A single XML file ***sarm.xml*** is generated upon the successful execution of the SARM Analyzer. The ***sarm.xml*** output file is stored in the current working directory. When running the SARM Analyzer, the full path of the SARM binary files has to be specified if the file is not in the same directory as the SARM Analyzer executable

SARM files are grouped based on the areas of instrumentation that is webserver (SWSE), Server thread, Siebel Web Engine, Workflow, Scripting Engine and Database Connector.

Example output schema:

```
<Group> *
  <Name>
  <ResponseTime>
    <Total>
    <Average>
    <NonRecursiveCalls>
    <RecursiveCalls>
    <Max>
    <Min>
```

## Siebel Application Response Measurement 7.5.3 Technical Note

```
<ExecutionTime>
  <Total>
  <Average>
  <Max>
  <Min>
<Parents>
  <ParentGroup> *
    <Name>
    <TotalContributingTime>
    <Calls>
    <Average>
    <ContributingTimePercent>
    <ContributingCallPercent>
<Children>
  <ChildGroup> *
    <Name>
    <TotalContributedTime>
    <Calls>
    <Average>
    <PercentageTime>
    <PercentageCalls>
```

Below is an explanation of each of the tags in the output schema:

- **Group**: refers to each area that is instrumented by SARM. Performance data is captured for the webserver (SWSE), Server threads, Database Connector, Scripting Engine, Workflow and Siebel Web Engine.
- **Response Time**: also called “inclusive time” in most commercial profiling tools. It is the time spent for a request between entering and exiting an instrumentation area.
  - a. **Total**: Total time spent on a request between entering and exiting an instrumentation area.
  - b. **Average**: average response time for a request. This is calculated by dividing total time (Total) by number of requests (NonRecursiveCalls).
  - c. **NonRecursiveCalls**: Number of times an instrumentation area is called. This tag helps identify the time it takes an instrumentation layer to respond to a request.
  - d. **RecursiveCalls**: One of the key features of the tool is the capability to handle recursion. For example, if a workflow step calls an Object Manager function, which also invokes another workflow step, then there is a recursion in workflow. Considering the number of times the workflow layer is being called, there are two relevant metrics: RecursiveCalls and NonRecursiveCalls. In this case, RecursiveCalls is 1 and NonRecursiveCalls is also 1. When calculating the response time, only the root-level call is being accounted for. When calculating execution time, both calls are being accounted.
  - e. **Max**: maximum time for a request between entering and exiting an instrumentation area.

## Siebel Application Response Measurement 7.5.3 Technical Note

- f. **Min:** minimum time for a request between entering and exiting an instrumentation area.
  
- **Execution Time:** It is often called “exclusive time” in most commercial profiling tools. It is the total time spent in a particular instrumentation area only, not including the time spent in the descendant layers.
  - a. **Total:** Total time spent on a request between entering and exiting an instrumentation area.
  - b. **Average:** average time spent on a request between entering and exiting an instrumentation area.
  - c. **Max:** maximum time for a request between entering and exiting an instrumentation area.
  - d. **Min:** minimum time for a request between entering and exiting an instrumentation area.
  
- **Parent:** parent of the group. This information helps identify the caller of a group and the total time and number of calls the group contributed to its parent’s response time.
  - a. **Name:** name of the parent group. A group is an area of instrumentation.
  - b. **Total Contributing Time:** total time a group contributed to the parent’s total time. For example, if SWSE calls the Object Manager (OM) and OM spends a total of 10 seconds, then the Total Contributing Time is 10. If Scripting Engine also calls OM, and OM spends 40 seconds when called by the Scripting Engine, then the Contributing Time Percentage of SWSE to OM is 20%. This is calculated as  $(\text{Total Contributing Time} / \text{Total OM Time}) * 100\%$  or  $((10/50) * 100\% = 20\%)$ . The Contributing Time of Scripting to OM in this case would be 80% or  $((40/50) * 100\% = 80\%)$ .
  - c. **Calls:** Number of times a parent group is called.
  - d. **Average:** average time spent in the parent.
  - e. **ContributingTimePercent:** time in percentage that this group contributed to the parent’s total time. For example WF contributed 25% to SWE (parent).
  - f. **ContributingCallPercent:** percentage of calls made from the parent to this group as a percentage of the total calls to all groups in a request. For example, WF is called 3 times in a request, SWE was called 30 times in the same request, and hence the contributing call percent of WF to SWE is 10%.
  
- **Children:** Children refer to the areas called by a parent group. A user can drill into a groups’ children information to determine response time break down within each of the child. By drilling down into the children’s information, the user can find potential performance bottlenecks.
  - a. **Name:** Name of the child group
  - b. **TotalContributedTime:** total time a child group contributed to the parent’s total response time. The sum of all children contribution’s time (response time) added to the area’s execution time should be the total response time for the area.
  - c. **Calls:** Number of calls made to a child group.
  - d. **Average:** average time spent on a child group.
  - e. **PercentageTime:** Percent of time spent on a child group divided by the time spent on all children’s groups.
  - f. **PercentageCalls:** Amount of calls in percentage that is spent on a child group.

# Siebel Application Response Measurement 7.5.3 Technical Note

## Example output

The data is displayed in nanoseconds as the sample was taken from a Unix machine.

```
- <xml>
  - <Group>
    <Name>SMI</Name>
    - <ResponseTime>
      <Total>625577844947</Total>
      <Average>1839422852</Average>
      <NonRecursiveCalls>177</NonRecursiveCalls>
      <RecursiveCalls>0</RecursiveCalls>
      <Max>133062957179</Max>
      <Min>3293465</Min>
    </ResponseTime>
    + <ExecutionTime>
      <Parents />
      + <Children>
    </Group>
  - <Group>
    <Name>Database</Name>
    - <ResponseTime>
      <Total>28846037763</Total>
      <Average>2804943</Average>
      <NonRecursiveCalls>10284</NonRecursiveCalls>
      <RecursiveCalls>106</RecursiveCalls>
      <Max>3623108101</Max>
      <Min>47397</Min>
    </ResponseTime>
    + <ExecutionTime>
      + <Parents>
      + <Children>
    </Group>
  - <Group>
    <Name>SarmIO</Name>
    - <ResponseTime>
      <Total>756465475</Total>
      <Average>6200536</Average>
      <NonRecursiveCalls>122</NonRecursiveCalls>
      <RecursiveCalls>0</RecursiveCalls>
      <Max>181488478</Max>
      <Min>730255</Min>
    </ResponseTime>
    + <ExecutionTime>
      + <Parents>
      <Children />
    </Group>
  - <Group>
    <Name>SWE</Name>
    - <ResponseTime>
      <Total>167202095979</Total>
      <Average>966486103</Average>
      <NonRecursiveCalls>173</NonRecursiveCalls>
      <RecursiveCalls>16</RecursiveCalls>
      <Max>51087996109</Max>
      <Min>141423</Min>
    </ResponseTime>
    + <ExecutionTime>
      + <Parents>
      + <Children>
    </Group>
  - <Group>
    <Name>Scripting Engine</Name>
    - <ResponseTime>
      <Total>42078467851</Total>
      <Average>825067997</Average>
      <NonRecursiveCalls>51</NonRecursiveCalls>
      <RecursiveCalls>0</RecursiveCalls>
```

# Siebel Application Response Measurement 7.5.3 Technical Note

```
        <Max>40459460508</Max>
        <Min>852767</Min>
    </ResponseTime>
    + <ExecutionTime>
    + <Parents>
    + <Children>
</Group>
- <Group>
  <Name>Workflow Engine</Name>
  - <ResponseTime>
    <Total>41809855132</Total>
    <Average>10452463783</Average>
    <NonRecursiveCalls>4</NonRecursiveCalls>
    <RecursiveCalls>14</RecursiveCalls>
    <Max>40450981149</Max>
    <Min>635413</Min>
  </ResponseTime>
  + <ExecutionTime>
  + <Parents>
  + <Children>
</Group>
</xml>
```

Figure 1

1. First, view all the groups and find out which one has the highest **ResponseTime**. By looking at figure 1 (XML file), note that Server Thread is the entry point to the Siebel server (doesn't have parent group) and it took **325** seconds.
  - a. Server Thread = 326 seconds (325,577,844,947 nanoseconds)
  - b. Database = 29 seconds (28,846,037,763)
  - c. SarmIO = .76 seconds (756,465,475)
  - d. SWE = 17 seconds (167,202,095,979)
  - e. Scripting Engine = 43 seconds (42,078,467,851)
  - f. Workflow Engine = 42 seconds (41,809,855,132)
2. When compared to the rest of the groups, the request spent most of its time on the Server Thread area; therefore the Server Thread information require further analysis. Also note the following information:
  - a. At an average, it took 1.8 seconds for Server Thread request to be processed.
  - b. The maximum time it took to process a Server Thread request was 133 seconds.
  - c. The minimum time it took to process a Server Thread request was .003 seconds.It is suspicious that a given Server Thread request took much longer than the average (133 vs. 1.8 seconds).
3. Next, look at the children group of the Server Thread and find out which child took the longest to processed (TotalContributedTime):

```
- <xml>
  - <Group>
    <Name>SMI</Name>
    + <ResponseTime>
  + <ExecutionTime>
  <Parents />
  - <Children>
    - <ChildGroup>
      <Name>Database</Name>
      <TotalContributedTime>10052385093</TotalContributedTime>
      <Calls>7378</Calls>
      <Average>1362481</Average>
      <PercentageTime>5.65</PercentageTime>
      <PercentageCalls>96.62</PercentageCalls>
```

## Siebel Application Response Measurement 7.5.3 Technical Note

```

</ChildGroup>
- <ChildGroup>
  <Name>SarmIO</Name>
  <TotalContributedTime>695242267</TotalContributedTime>
  <Calls>85</Calls>
  <Average>8179320</Average>
  <PercentageTime>0.39</PercentageTime>
  <PercentageCalls>1.11</PercentageCalls>
</ChildGroup>
- <ChildGroup>
  <Name>SWE</Name>
  <TotalContributedTime>167202095979</TotalContributedTime>
  <Calls>173</Calls>
  <Average>966486103</Average>
  <PercentageTime>93.96</PercentageTime>
  <PercentageCalls>2.27</PercentageCalls>
</ChildGroup>
</Children>
</Group>
+ <Group>
+ <Group>
+ <Group>
+ <Group>
+ <Group>
</xml>

```

**Figure 6**

- a. From Figure 6, SWE's contribution time was the highest with 167 seconds vs. 10 seconds for Database and .6 seconds for SarmIO.
- b. Note that of the total number of calls spent on the children groups, only 2.27% of the calls were made to SWE (SWE Calls / (Database Calls + SarmIO Calls + SWE Calls)) \*100% or (173 / (7378+85+173))\*100%.

However, even though only 2.27% of the calls were made to SWE, those calls accounted for 93.96% of the response time within the children's group.

$(\text{SWE TotalContributedTime} / (\text{Database TotalContributedTime} + \text{SarmIO TotalContributedTime} + \text{SWE TotalContributedTime})) * 100\%$

or  $(167202095979 / (10052385093 + 695242267 + 167202095979)) * 100\%$

These findings further indicate that there are very few calls within the SWE child group (173), but the percent of time spent on those SWE calls were very high (93.96%). Therefore additional analysis should be done on the SWE group to isolate the performance problem.

4. Look at the SWE group in more detail and specifically expand the children's groups.

```

- <xml>
+ <Group>
+ <Group>
+ <Group>
- <Group>
  <Name>SWE</Name>
- <ResponseTime>
  <Total>167202095979</Total>
  <Average>966486103</Average>
  <NonRecursiveCalls>173</NonRecursiveCalls>
  <RecursiveCalls>16</RecursiveCalls>
  <Max>51087996109</Max>
  <Min>141423</Min>
  </ResponseTime>
- <ExecutionTime>
  <Total>173968409607</Total>
  <Calls>189</Calls>

```



## Siebel Application Response Measurement 7.5.3 Technical Note

$$(167,202,095,979 - (18,540,207,197 + 55,622,335 + 34,393,218,236 + 42,078,467,851 + 375,294,777)) = 71,759,285,583$$

**Note:** In the example above (Figure 7) and in the current 7.5.3 version of SARM, the parent's execution time is not displayed correctly. In order to calculate the parent's execution time, subtract the children's total contribution time from the parent's total response time.

- b. Identifying the percentage of time a child group contributed to the parent's total response time helps to illustrate which child contributed the most to the parent's total response time. This information helps in identifying if additional investigation needs to be done on a particular child.

Child Area	Total contributed Time	% of SWE's Response Time spent on child area
Database	18,540,207,197	11.00%  ((DB TotalContributedTime/SWE Total Response Time) *100%) ((18,540,207,197/167,202,095,979) *100%) = 11.00%
SarmIO	55,622,335	0.03%  ((SarmIO TotalContributedTime/SWE Total Response Time) *100%) ((55,622,335/167,202,095,979) *100%) = 0.03%
SWE	34,393,218,236	<b>20.50%</b>  ((SWE TotalContributedTime/SWE Total Response Time) *100%) ((34,393,218,236/167,202,095,979) *100%) = 20.50%
Scripting Engine	42,078,467,851	<b>25.10%</b>  ((Scripting Engine TotalContributedTime/SWE Total Response Time) *100%) ((42,078,467,851/167,202,095,979) *100%) = 25.10%
Workflow	375,294,777	0.23%  ((Workflow TotalContributedTime/SWE Total Response Time) *100%) ((375,294,777/167,202,095,979) *100%) = 0.23%
% of SWE Response Time spent on its children	95,442,810,396	56.86%  (11.00% + 0.03% + 20.50% + 25.10% + 0.23%) = 56.86%
SWE's Execution Time	71,759,285,583	43.14%

**Table 2**

- a. From Figure 7 and Table 2, the total contribution time for each of the children's areas within the SWE group are:

Database = 19 seconds (18,540,207,197) see underlined numbers

## Siebel Application Response Measurement 7.5.3 Technical Note

SarmIO = .06 seconds (55,622,335) see underlined numbers

SWE = 34 seconds (34,393,218,236) see underlined numbers

Scripting Engine = 42 seconds (42,078,467,851) see underlined numbers

By looking at Table 2, most of the time used by SWE was consumed by the Scripting Engine (42 seconds)

- b. 25% of the SWE time was spent on the Scripting Engine (Scripting Engine TotalContributedTime / SWE ResponseTime) \*100%) or

(42,078,467,851 / 167,202,095,979) \*100%). This should be flagged as an area that should be further investigated.

SWE TotalContributedTime = 167,202,095,979

Scripting Engine TotalContributedTime = 42,078,467,851

20.5% of the SWE time was spent on SWE itself, however, there is no additional performance data provided for SWE; therefore additional analysis on the Scripting Engine area is needed to isolate the problem.

5. Next, look at the Scripting Engine group.

```
- <xml>
+ <Group>
+ <Group>
+ <Group>
+ <Group>
- <Group>
  <Name>Scripting Engine</Name>
  - <ResponseTime>
    <Total>42078467851</Total>
    <Average>825067997</Average>
    <NonRecursiveCalls>51</NonRecursiveCalls>
    <RecursiveCalls>0</RecursiveCalls>
    <Max>40459460508</Max>
    <Min>852767</Min>
  </ResponseTime>
  - <ExecutionTime>
    <Total>643907496</Total>
    <Calls>51</Calls>
    <Average>12625637</Average>
    <Max>564757316</Max>
    <Min>852767</Min>
  </ExecutionTime>
  - <Parents>
    - <ParentGroup>
      <Name>SWE</Name>
      <TotalContributingTime>42078467851</TotalContributingTime>
      <Calls>51</Calls>
      <Average>825067997</Average>
      <ContributingTimePercent>100.00</ContributingTimePercent>
      <ContributingCallPercent>100.00</ContributingCallPercent>
    </ParentGroup>
  </Parents>
  - <Children>
    - <ChildGroup>
      <Name>Workflow Engine</Name>
      <TotalContributedTime>41434560355</TotalContributedTime>
      <Calls>2</Calls>
      <Average>20717280177</Average>
      <PercentageTime>100.00</PercentageTime>
      <PercentageCalls>100.00</PercentageCalls>
    </ChildGroup>
  </Children>
</Group>
+ <Group>
```

# Siebel Application Response Measurement 7.5.3 Technical Note

</xml>

Figure 8

- a. Notice that the maximum response time is around 40 seconds whereas the minimum response time is 0.00085 seconds, and the average is 0.825 seconds. The maximum and minimum response times are very far apart, and probably the maximum response time value is making the average number look much higher. If the maximum response time is subtracted from the total and the average calculated, a more realistic average time is derived.

(Total Response Time – Max)/NonRecursiveCalls = Average

(42,078,467,851 – 40,459,460,508)/50 = 32,380,146 (0.032 seconds)

- b. By calculating this new average, it is apparent that on an average, the execution of the scripts was efficient. It should be noted that these numbers should be compared to a base line and are always relative.
- c. It is also noticeable that one of the scripts took a very long time. By looking at the file, the name of the script is not known, but more investigation can occur by looking at the CSV file (look for CSV conversion for additional explanation).

## Call Map Generation

Using the –xml switch with the SARM Analyzer will produce a more detailed call map of the transactions captured by SARM.

### Command Syntax:

On Windows: SARMAalyzer –xml <sarm file> <sarm file.xml>

On Unix: sarmanalyzer –xml <sarm file> <sarm file.xml>

**Output File Name:** <sarm file>.xml

### Example on Windows:

SARMAalyzer.exe –xml S01\_P20862\_N0002.sarm > S01\_P20862\_N0002.sarm.xml

### Description:

For a given a SARM file, the SARM Analyzer constructs a map with all the call references. Each node in the call map represents an instrumentation instance. Using this option generates an XML file containing all the calls made by each component monitored.

Example output:

The following is the data for each node.

```
<Node>
  <SarmID>
  <ParentID>
  <RootID>
  <SenderSrvID>
  <SenderProcID>
  <Area>
  <SubArea>
  <StartTime>
  <Duration>
  <UserInt1>
  <UserInt2>
```

# Siebel Application Response Measurement 7.5.3

## Technical Note

<UserString>

<Node>\*

Below is an explanation of each of the tags in the output schema:

- **Node**: Each instance of an instrumented area is a node. Each node can have zero to many nodes as its descendants
- **SarmID**: A unique number representing a SARM instrumentation point.
- **ParentID**: A unique number representing the caller of an instrumentation point within the same request. The caller is another instrumented area.
- **RootID**: A unique number assigned to a request submitted from the SWSE to the Siebel Server. RootID is also known as Request ID.
- **SenderSrvID**: Reserved for future use.
- **SenderProcID**: Reserved for future use.
- **Area**: Instrumentation element within the Siebel architecture. The seven elements that have been instrumented to collect response time information are: SarmIO, SWSE, Server Thread, SWE, Workflow, Scripting Engine and Database Connector.
- **SubArea**: Detailed instrumentation within an area of the Siebel architecture. For example, SARM will capture response timing for invoking a method (Invoke Method) or executing a step (Step Execution) within a Workflow execution.
- **StartTime**: Internal representation of the SARM record timestamp.
- **Duration**: Total time to execute the instrumented area.
- **UserInt1**: Context information captured at the point of instrumentation. The value depends on the instrumented area.
- **UserInt2**: Context information captured at the point of instrumentation. The value depends on the instrumented area.
- **UserString**: Context information captured at the point of instrumentation. The value depends on the instrumented area.

### Example output

The data is displayed in nanoseconds as the sample was taken from a Unix machine.

```
- <xml>
+ <Node>
- <Node>
  <SarmID>3562</SarmID>
  <ParentID>-33</ParentID>
  <RootID>3562</RootID>
  <SenderSrvID>-11</SenderSrvID>
  <SenderProcID>-22</SenderProcID>
  <Area>4</Area>
  <SubArea>3</SubArea>
  <StartTime>1057722340554572519</StartTime>
  <Duration>149747</Duration>
  <UserInt1>0</UserInt1>
  <UserInt2>0</UserInt2>
  <UserString />
</Node>
+ <Node>
+ <Node>
```

# Siebel Application Response Measurement 7.5.3

## Technical Note

**Figure 9**

- a. Figure 9 shows a sample section of the SARM call graph information. Notice that the SarmID and RootID share the same value of 3562. When the SarmID and the RootID are the same, then this node is the root of the request.
- b. Area 4 represents the Database Connector area, and SubArea 3 indicated the time to fetch a database record. In this scenario, the total time to fetch the next record from the database was 149747 nanoseconds.

### User Session trace

A user session trace can be created by using the SARM analyzer with the `-w` switch and by providing SARM binary files from the web server and the application server. Additionally, the `-csv` `-prop` switches can be used in combination to create another session trace file with more detail information.

To execute a User Session Trace, run the SARM Analyzer in two different modes:

#### Command Syntax:

On Windows: `SARMAalyzer -w <websrvr sarm file> -s <siebsrvr sarm file> -u <user name>`

On Unix: `sarmanalyzer -w <websrvr sarm file> -s <siebsrvr sarm file> -u <user name>`

**Output File Name:** `<user name>.xml`

#### Example on Windows:

`SARMAalyzer -w S01_P20862_N0002.sarm -s S01_P14268_N0000.sarm -u sadmin.xml`

#### Command Syntax:

On Windows: `SARMAalyzer.exe -csv -prop <websrvr sarm file> > <some_file_name.csv>`

On Unix: `sarmanalyzer.exe -csv -prop <websrvr sarm file> > <some_file_name.csv>`

**Output File Name:** `<some_file_name.csv>`

#### Example on Windows:

`SARMAalyzer.exe -csv -prop S01_P20862_N0002.sarm > S01_P20862_N0002_SWSE.csv`

### Description

SARM Analyzer expects one SARM binary file from the SWSE on the Web Server, and one SARM binary file from the Siebel Server.

The XML output file contains detailed information on each of the SWSE requests that the user has made. If the user has logged into the system multiple times, then the output will show that there are multiple sessions. The SWSE requests are grouped into specific login sessions, and sorted by the time the requests were made.

#### Example Output:

```
<Session>
  <SessionID>
  <LoginName>
  <SWERequests>
    <SWERequest>*
      <ReqID>
      <ClickID>
```

# Siebel Application Response Measurement 7.5.3

## Technical Note

<ReqTimeStamp>  
<RequestBody>  
<TotalServerTime>  
<WebServerTime>  
<InfraTime>  
<SiebSvrTime>  
<DatabaseTime>  
<DatabaseCalls>  
<SiebsvrDetail>  
    <Group>\*  
<Name>  
<ResponseTime>  
    <Total>  
    <Average>  
    <NonRecursiveCalls>  
    <RecursiveCalls>  
    <Max>  
    <Min>  
<ExecutionTime>  
    <Total>  
    <Average>  
    <Max>  
    <Min>  
<Parents>  
    <ParentGroup> \*  
        <Name>  
        <TotalContributingTime>  
        <Calls>  
        <Average>  
        <ContributingTimePercent>  
        <ContributingCallPercent>  
<Children>  
    <ChildGroup> \*  
        <Name>  
        <TotalContributedTime>  
        <Calls>  
        <Average>  
        <PercentageTime>  
        <PercentageCalls>

- **Session**: refers to a specific user session

## Siebel Application Response Measurement 7.5.3 Technical Note

- **SessionID**: Refers to a unique user session id made up of hexadecimal values. Some session ids are blank because they represent anonymous user sessions. The first component of the Session Id refers to the Server ID (currently a constant number reserved for future use), the second refers to the Operating System Process ID and the last section is the Siebel Task ID. To process id and task id needs to be converted from hexadecimal to decimal.

**!1.2b40.182b**

Server ID = !1

Process ID = 2b40 Represents the Operating System Process ID 11072

Task ID = 182b Represents the Siebel Task ID 6187

- **LoginName**: Login name of the user whose session is being investigated.
- **ReqID**: RequestID: Incremental numeric value to count the number of requests.
- **ClickID**: Used to associate multiple requests to a single user action, such as 'click on a button'.
- **RequestBody**: Total time to handle a request from the web server to the Siebel server. Detail timing is grouped by web server, infrastructure, Siebel server and database time.
- **TotalServerTime**: total time on the server (includes web server, Siebel server, and network time).
- **WebServerTime**: total time spent on the web server for a given request.
- **InfraTime**: Infrastructure Time. Total time spent between the web server and the Siebel server. This may also include some Siebel infrastructure time routing the request to the handling Siebel server task
- **ReqTimeStamp**: Request Time Stamp. This is the time when the request was made.
- **SiebSrvrTime**: Total time spent on the Siebel server.
- **DatabaseTime**: Total database Time. Database time includes the time spent on the network when communicating to the database.
- **DatabaseCalls**: Database Calls. Number of calls to the Siebel server database connector layer.
- **SiebsrvrDetail**: Total time by instrumentation area within the Siebel server. The areas of instrumentation within the Siebel server are: server thread, SWE, workflow, scripting engine and database connector.

### Example output

The data is displayed in nanoseconds as the Sample originates from a Unix machine.

## Siebel Application Response Measurement 7.5.3 Technical Note

```
- <xml>
- <Session>
  <SessionID />
  <LoginTime>Mon Jul 7 21:56:53 2003</LoginTime>
  <SWERequests />
</Session>
+ <Session>
- <Session>
  <SessionID>!1.2b40.182b</SessionID>
  <LoginTime>Mon Jul 7 22:40:25 2003</LoginTime>
  + <SWERequests>
</Session>
- <Session>
  <SessionID />
  <LoginTime>Mon Jul 7 22:40:32 2003</LoginTime>
  <SWERequests />
</Session>
- <Session>
  <SessionID>!1.2b40.182d</SessionID>
  <LoginTime>Mon Jul 7 22:43:12 2003</LoginTime>
  + <SWERequests>
</Session>
- <Session>
  <SessionID>!1.2b40.182f</SessionID>
  <LoginTime>Mon Jul 7 23:28:41 2003</LoginTime>
  + <SWERequests>
</Session>
- <Session>
  <SessionID>!1.2b40.1830</SessionID>
  <LoginTime>Mon Jul 7 23:28:47 2003</LoginTime>
  + <SWERequests>
</Session>
- <Session>
  <SessionID>!1.2b40.1831</SessionID>
  <LoginTime>Mon Jul 7 23:34:57 2003</LoginTime>
  + <SWERequests>
</Session>
</xml>
```

Figure 10

1. By looking at Figure 10, notice that there are multiple sessions for the same user.
2. To run session tracing, both the web server file and the Siebel Server file are used. However, there may be multiple SARM files on the server (one set of SARM files per process). To identify which SARM file to use to correlate SWE requests with server requests to construct a user's session trace, follow the instructions below:
  - a. Concatenate the web server SARM files.

In a Windows environment, use any utility to concatenate the files.

In a Unix environment, use the following command:

```
cat <list of files> >> <filename.sarm>
```

Example

```
cat S01_P11072_N0001.sarm >> sum.sarm
cat S01_P11072_N0003.sarm >> sum.sarm
cat S01_P11072_N0004.sarm >> sum.sarm
```

## Siebel Application Response Measurement 7.5.3 Technical Note

- b. Run the following command **SARMAalyzer –csv –prop - <web server sarm file> > web.csv** to get a CSV formatted SARM SWSE File. Within the SWSE Property File, look for the SARM ID of the username that requires analysis. The username will be found in the PropVal column. If the user has logged in multiple times, look at the timestamp, also found in the PropVal column.

	A	B	C	D
1	SarmID	PropID	RootID	PropVal
2	2	1	1	sadmin
3	2	3	1	!1.2b40.181b
4	2	2	1	Mon Jul 7 21:55:05 2003
5	1	4	1	
6	4	1	3	sadmin
7	4	3	3	!1.2b40.181c
8	4	2	3	Mon Jul 7 21:55:05 2003
9	3	4	3	
10	1495	4	1495	
11	1574	4	1574	
12	1576	1	1574	sadmin
13	1576	3	1574	
14	1576	2	1574	Mon Jul 7 21:56:53 2003
15	1578	4	1578	
16	1580	1	1578	bcook
17	1580	3	1578	
18	1580	2	1578	Mon Jul 7 21:57:08 2003
19	1582	4	1582	
20	1584	4	1584	
21	1587	1	1584	sadmin
22	1587	3	1584	
23	1587	2	1584	Mon Jul 7 22:02:54 2003
24	1589	4	1589	
25	1592	1	1589	sadmin
26	1592	3	1589	!1.2b40.182b
27	1592	2	1589	Mon Jul 7 22:40:25 2003

**Table 3**

- c. Look for all the records that have the property value of the username, in our case “sadmin”. Right below the sadmin property name is the session of the user in hexadecimal. This value needs to be converted to decimal.

**!1.2b40.182b**

Server ID = !1

Process ID = 2b40

Task ID = 182b

- d. Take the second piece of information such as !1.**2b40**.182b and convert it to decimal (11072). The Process ID 11072 will correspond to the Siebel Server Process ID (SARM file). Here it shows the OM Process ID for the user that is having problems.
- e. Run the user session trace **SARMAalyzer –w <websrvr sarm file> -s <siebsrvr sarm file> -u <username>** to trace all the requests the user has made and identify potential performance problems using the process id identified in the previous step.
- f. Open the output file from the User Trace (sadmin.xml). Find out the time user was logged in when the problem occurred the appropriate session Id to investigate can be obtained. Usually the user will not have logged in too many times (as in the case of

## Siebel Application Response Measurement 7.5.3 Technical Note

sadmin) so the correct session is more quickly identified (the user may not remember specific login times).

- g. For that given session, identify which request took the longest time (TotalServerTime). Drill into its details to identify which of the groups within that request took the longest time. There are many requests in a given session, therefore it is recommended to write additional tools to easily identify the highest response time request. The user needs to keep in mind that the highest response time is always relative to the base line.

```
- <xml>
+ <Session>
+ <Session>
+ <Session>
+ <Session>
- <Session>
  <SessionID>1.2b40.182d</SessionID>
  <LoginTime>Mon Jul 7 22:43:12 2003</LoginTime>
- <SWERequests>
  + <SWERequest>
  + <SWERequest>
  + <SWERequest>
  + <SWERequest>
  - <SWERequest>
    <ReqID>4</ReqID>
    <ClickID>-1</ClickID>
    <RequestBody />
    <TotalServerTime>39905869762</TotalServerTime>
    <WebServerTime>4739348</WebServerTime>
    <InfraTime>575600</InfraTime>
    <SiebServerTime>28290089242</SiebServerTime>
    <DataBaseTime>11610465572</DataBaseTime>
    <DataBaseCalls>1446</DataBaseCalls>
  + <SiebsrvrDetail>
  </SWERequest>
```

Figure 11

- h. Use a similar process to the aggregation analysis to identify which area in the Siebel architecture was high in performance. Additional analysis would be needed using the CSV file to identify specific sub-areas are the problem areas.

# Siebel Application Response Measurement 7.5.3 Technical Note

## SARM Binary File to CSV Conversion

### Command Syntax:

On Windows: SARManalyzer.exe -csv -sarm <sarm file name> > <some\_file\_name.csv>

On Unix: sarmanalyzer.exe -csv -sarm <sarm file name> > <some\_file\_name.csv>

**Output File Name:** <some\_file\_name>.csv

**Example on Windows:** SARManalyzer.exe -csv -sarm S01\_P20862\_N0002.sarm > S01\_P20862\_N0002.csv

### Description:

Use this command to convert the binary file into a CSV without any interpretation or aggregation.

**UserInt1, UserInt2, UserStr:** context information captured at the point of instrumentation. The information will depend on the area of instrumentation. For example, UserStr can contain a business component name, a workflow name, or a view name. The Area and SubArea values will provide context to what the UserStr value means.

### Example output

1. Start by deleting the comments generated during the creation of the CSV file (Exporting data to CSV. Please wait...).
2. When using Excel, sort the CSV file by the ID column and use the auto filter feature to filter the **Top 10** requests by **Duration**. Each of the **Areas** and **Sub-areas** corresponds to one of the instrumentation points in the architecture (Refer to Table 1). For example, scripting engine is denoted by area 5.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	ID	ParentID	RootID	SenderID	SenderID	Area	SubArea	StartTime	Duration (ns)	Duration	UserInt1	UserInt2	UserStr
2	1	-33	1	-11	-22	3	1	1.06E+18	133,063,000,000	133.063	7194	3	
7762	7771	-33	7771	-11	-22	3	1	1.06E+18	12,787,483,831	12.787	7207	15	
9014	9056	-33	9056	-11	-22	3	1	1.06E+18	51,095,897,268	51.096	7207	36	
9015	9057	9056	9056	-11	-22	8	1	1.06E+18	51,087,996,109	51.088	0	0	InvokeMethod
9857	10400	-33	10400	-11	-22	3	1	1.06E+18	40,525,465,841	40.525	7207	63	
9858	10401	10400	10400	-11	-22	8	1	1.06E+18	40,518,945,027	40.519	0	0	InvokeMethod
9859	10403	10401	10400	-11	-22	5	3	1.06E+18	40,459,460,508	40.459	0	0	BusComp_PreSetF
9860	10404	10403	10400	-11	-22	7	1	1.06E+18	40,450,981,149	40.451	0	0	RunProcess
9940	10498	10404	10400	-11	-22	7	2	1.06E+18	40,135,986,509	40.136	0	0	ABC SARM Test
10324	11056	-33	11056	-11	-22	3	1	1.06E+18	12,614,397,962	12.614	7207	89	

**Table 4**

- a. From Table 4, the following information should be highlighted:
  - SERVER THREAD (area =3) took 133 seconds. Note that the **ID** and **RootID** have the same number of 1. A RootID of 1 represents the starting points of all requests. The first SERVER THREAD operation having a high response time does not appear too suspicious since it could be attributed to the system warming and caching information for subsequent requests.
  - Server Thread (SMI), SWE, Script Engine, SarmIO, SWE Plug-in took nearly 40 seconds. All of these areas have the same **RootID** of **10400**, which

## Siebel Application Response Measurement 7.5.3 Technical Note

means that they are part of the same request. This request should be further investigated.

3. Put a filter on the **RootID** column for **10400**. This will yield all the instrumentation points executed as a result of a single request.
4. It is acceptable to leave the “Top 10” filter on the **Duration** column. Although not required it will help reduce the number of operations to be reviewed. The result will be similar to that shown on Table 5.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	ParentID	ID	RootID	Send	Send	Area	SubArea	StartTin	Duration (ns)	Duration	UserInt'	UserInt'	UserStr		
9857	-33	10400	10400	-11	-22	3	1	1.06E+18	40,525,465,841	40.525	7207	63			
9858	10400	10401	10400	-11	-22	8	1	1.06E+18	40,518,945,027	40.519	0	0	InvokeMethod		
9859	10401	10403	10400	-11	-22	5	3	1.06E+18	40,459,460,508	40.459	0	0	BusComp_PreSetFieldValu		
9860	10403	10404	10400	-11	-22	7	1	1.06E+18	40,450,981,149	40.451	0	0	RunProcess		
9940	10404	10498	10400	-11	-22	7	2	1.06E+18	40,135,986,509	40.136	0	0	ABC SARM Test		

**Table 5**

- a. From Table 5, notice that line 9857 has an **ID** and **RootID** of **10400**, which means that this is the starting point of the request. In this case, Server Thread (area =3) was the first area of instrumentation for this request.
- b. Notice the values of the **ParentID** and **ID** columns and walk through the call hierarchy:
  1. 10400 calls 10401
  2. 10401 calls 10403
  3. 10403 calls 10404
  4. 10404 calls 10498
- c. Now examine the values of the Area and SubArea to find out which Siebel infrastructure areas were called:
  1. 10400 calls 10401 Server Thread:Request Handling (Area 3, SubArea 1) called  
SWE:Process SWE Command (Area 8, SubArea 1)
  2. 10401 calls 10403 SWE:Process SWE Command (Area 8, SubArea 1) called  
Scripting Engine:eScript Execution (Area 5, SubArea 3)
  3. 10403 calls 10404 Scripting Engine:eScript Execution (Area 5, SubArea 3) called  
Workflow:Invoke Method (Area 7, SubArea 1)
  4. 10404 calls 10498 Workflow:Invoke Method (Area 7, SubArea 1) called  
Workflow:Process Init (Area 7, SubArea 2)
- d. In addition, the objects that were called by looking at the **UserString** column (M) are identified. For example ID 10403 calls the “**BusComp\_PreSetFieldValue**” eScript code and ID 10498 calls the “**ABC SARM Test**” workflow.
- e. Notice that the **Duration** for each of the operations is over 40 seconds. This is because the parent operation includes the time of its children. Therefore each

## Siebel Application Response Measurement 7.5.3 Technical Note

level up the hierarchy will be greater. Conversely when traversing down the hierarchy each time value will be lower.

- f. Because operation 10498 (line 9940) takes 40.136 seconds, it is evident that the problem resides somewhere in the "ABC SARM Test" Workflow, which require further analysis.

5. Now examine the operation **ID** 10498 in more detail.

- a. Remove all the filters
- b. Add a filter on the **ParentID** column for 10498

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	ParentID	ID	RootID	Sen	Send	Area	SubArea	StartTin	Duration (ns)	Duration	UserInt'	UserInt2	UserStr
9941	10498	10499	10400	-11	-22	7	4	1.06E+18	103,600	0	0	0	Start
9942	10498	10505	10400	-11	-22	7	4	1.06E+18	358,825	0	0	0	Check i
9943	10498	10517	10400	-11	-22	7	4	1.06E+18	10,044,373,913	10.044	0	0	Wait
9956	10498	10537	10400	-11	-22	7	4	1.06E+18	63,011,411	0.063	0	0	Iterate i
9957	10498	10548	10400	-11	-22	7	4	1.06E+18	201,715	0	0	0	Check i
9958	10498	10560	10400	-11	-22	7	4	1.06E+18	10,006,015,419	10.006	0	0	Wait
9959	10498	10568	10400	-11	-22	7	4	1.06E+18	1,179,592	0.001	0	0	Iterate i
9960	10498	10579	10400	-11	-22	7	4	1.06E+18	197,241	0	0	0	Check i
9961	10498	10591	10400	-11	-22	7	4	1.06E+18	10,008,038,683	10.008	0	0	Wait
9962	10498	10599	10400	-11	-22	7	4	1.06E+18	1,173,722	0.001	0	0	Iterate i
9963	10498	10610	10400	-11	-22	7	4	1.06E+18	199,984	0	0	0	Check i
9964	10498	10622	10400	-11	-22	7	4	1.06E+18	10,008,109,082	10.008	0	0	Wait
9965	10498	10630	10400	-11	-22	7	4	1.06E+18	36,618	0	0	0	End

**Table 6**

- c. Table 6 shows each instrumentation point within Workflow.
- d. By looking at line 9943, 9958, 9961 and 9964, it is evident that there were 10 seconds delays in each **Wait** step and the same step was called four times. This is the root cause of the performance problem.

**Note:** Improperly placed wait times are a common cause of workflow performance problems.

By using the data provided from the aggregation analysis in conjunction with data from the CSV file, a user can view response time information for individual elements within a request and immediately understand and isolate the location of the problem.

# Siebel Application Response Measurement 7.5.3

## Technical Note

### Best Practices

Below is a set of recommendation when using SARM.

When carrying out performance analysis, a baseline should be used. Siebel recommends that administrators take a snapshot of their system at a given point in time. This snapshot will be used to compare system behavior and identify performance problems. Administrators can also write additional tools to filter, aggregate, and compute the data, to help diagnose any potential problems.

The performance aggregation data is useful for diagnosing performance data at a given point in time. The administrator can look at the data by group and diagnose the area that is resulting in poor performance. After having a high level view of the performance data, more detailed information can be extrapolated by running the '**SARMAAnalyzer -CSV**' option and looking at the details of each area to identify the root of the problem. The performance aggregation data is also useful for doing trend analysis over a period of time.

As discussed, at most there will be four SARM files per process. The correct SARM file will depend on what the administrator is looking for. If the administrator wants to see the performance aggregation data for a given point in time, then SARM Analyzer should be run against a single SARM file (Note that if the maximum memory size (buffer size) is too small, the time stamp of the SARM files may be very close in time). If on the other hand the administrator wants to see the data for multiple requests for a given process, then it is recommended to concatenate the SARM files into a single file and run SARMAAnalyzer against that single file.

There may be situations where SARM shows that most of the time spent in a request takes place within SWE. Although SARM does not provide further detail on SWE, the administrator should check the complexity of the Siebel Views in the application. One way to identify complex or non-performant Siebel Views is to perform the following steps:

1. Identify which views appear to be slow based on user feedback or performance and scalability testing
2. Define a usage scenario that involved calling the slow view(s)
3. Enable SARM
4. Modify the configuration of the slow view(s)
5. Run SARM Analyzer to get the output described in this section
6. If the SWE time increases or decreases, how much time the configuration affects performance can be seen directly.
7. If the feedback came from a user, run the Session Trace described on this paper for more precise validation.

It is recommended to monitor performance activity during the testing and user acceptance stage to detect incipient problems before they have an adverse effect. In some cases, when SARM is used to monitor performance continuously, the user may detect a situation that requires additional data collection or explicit recreation of the problem to collect additional analysis data. Further analysis of the SARM data can be done by writing additional post-processing tools.

### Summary

SARM provides performance metrics such as cumulative and average response times, enabling the user to diagnose response time information in the Siebel architecture. SARM captures response time information based on different areas in the Siebel architecture; showing the actual

## Siebel Application Response Measurement 7.5.3 Technical Note

server processes and flow of data to immediately get in-depth response time information to quickly diagnose and analyze the Siebel solution.

For more information on SARM and the ARM standard, please refer to:

Siebel 7.5.3 Bookshelf: Performance and Tuning Guide

<http://www.opengroup.org/management/arm.htm>

<http://regions.cmg.org/regions/cmgarml/index.html>