Paper 009-2013

MACUMBA: Modern SAS® GUI Debugging Made Easy

Michael Weiss, Bayer Pharma AG, Berlin, Germany

ABSTRACT

MACUMBA is an in-house-developed application for SAS® programming. It combines interactive development features of PC-SAS, the possibility of a client-server environment and unique state-of-the-art features that were always missing. This presentation covers some of the unique features that are related to SAS code debugging. At the beginning, special code execution modes are discussed. Afterwards, an overview of the graphical implementation of the single-step debugger for SAS macros and DATA step is provided. Additionally, the main pitfalls of development are discussed.

INTRODUCTION

The MACUMBA development was started in 2007 as a graphical implementation of the PC-SAS DATA step debugger. Step By Step (beside the daily work) lots of other features where added to support SAS program and macro development, data examinations and a lot more. Figure 1 shows a basic overview of the application design.

MACUMBA is implemented in pure Java and uses the SAS IOM technology to access a SAS Object Spawner for “interactive like” SAS code execution and data access.

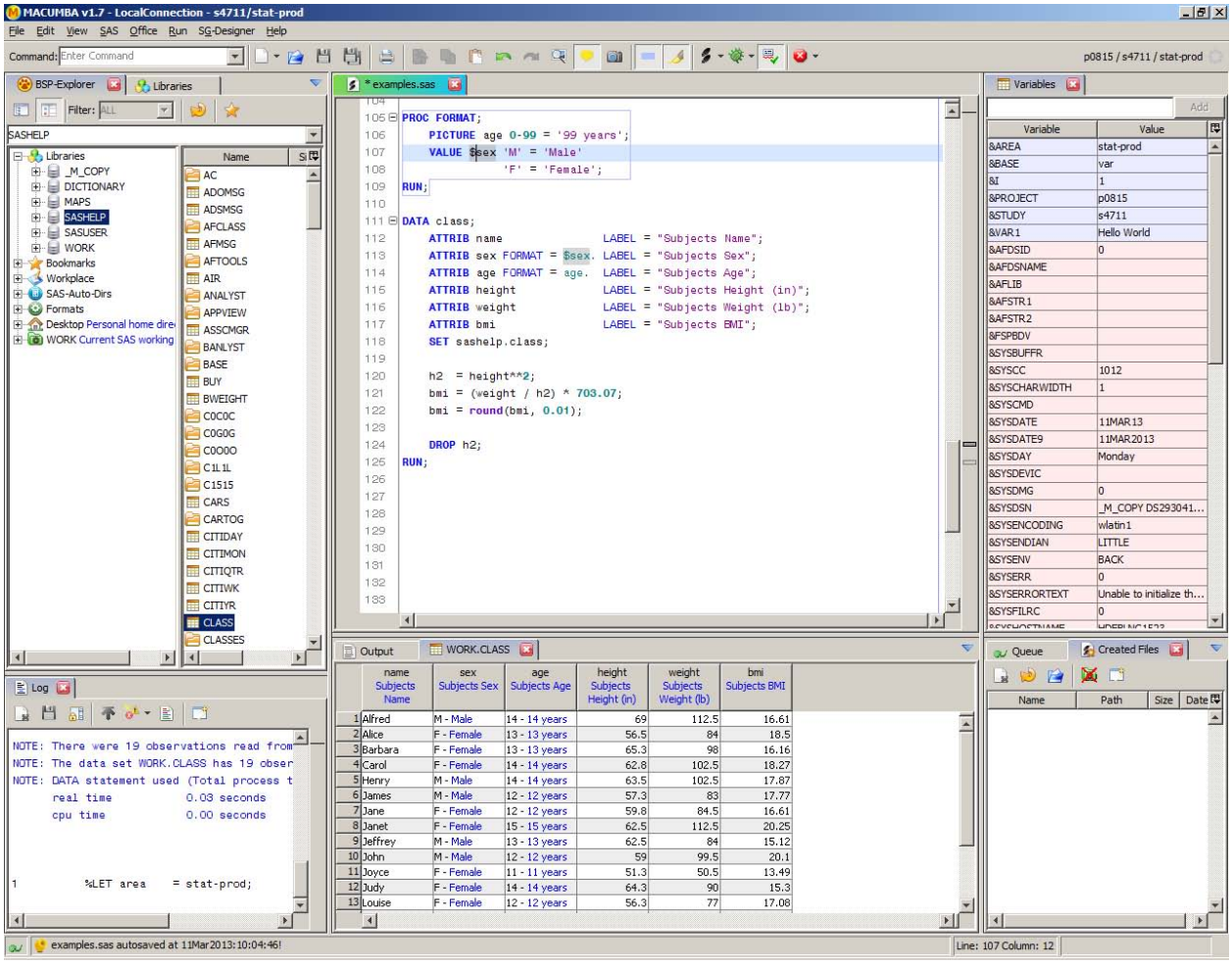

Figure 1: MACUMBA Main Window

SPECIAL CODE EXECUTION MODES

Some of the “easy to implement” features of MACUMBA are the special code execution modes. As most of the IDE’s for interpreted languages, PC-SAS provides the possibility to submit either the complete program code to the SAS interpreter or only the currently selected code. Especially within program development some special execution modes would be useful.

The following additional execution modes are provided by MACUMBA and explained in this paper:

- “Run To Line”, “Run From Line” and “Run Step” – Execute a specific part of the program
- “Run in Template” – Allows to define a custom code template that encloses the executed code
- “Run As Macro” – Allows to execute some code lines as a temporary SAS macro
- “Resolve Code” – Resolves all macro code and provides the real executed SAS code by using MPRINT

“RUN TO LINE”, “RUN FROM LINE” – THE BORDER PATROL

Depending on the program design it is sometimes necessary to execute a program from the beginning to the current position, e.g. to see the consequences of a change. In large programs this would require to scroll up, mark the beginning of the program, scroll down again, search the correct line and select the complete block of code for execution. In other situations it might be necessary to execute the program from the current position until the end of the program. This also would require a significant amount of scrolling in large programs.

MACUMBA provides the following two special execution modes to handle this:

- “RUN TO LINE” – Executes the code between start of the program and the current line.
- “RUN FROM LINE” – Executes the code from the current line until the program end.

“RUN STEP” – A SMART EXECUTION HELPER

In addition to “RUN TO LINE” and “RUN FROM LINE” Macumba provides another execution mode, the execution of a single step. Such a step could be a DATA- or PROC-Step, a global COMMAND (e.g. FILENAME, LIBNAME or OPTIONS) or a macro call or definition.

Especially when working on a big DATA step it is often inconvenient scrolling up and down to select the DATA step for execution.
In MACUMBA this special execution mode is known as “RUN STEP”. When invoking RUN STEP a dialog window is shown (Figure 2) that provides a list of possible steps to execute. For example, if invoked inside of a DATA step, the entry “DATA-Step” is available. If invoked inside of a macro definition the “%MACRO <name> Definition” entry is available and so on. This provides a simple way to execute a code part without the need to correctly select it.

As a special feature an SQL processing is implemented that allows executing a single SQL statement from inside of a SQL procedure block. In this case the statement is automatically surrounded by PROC SQL; <…> QUIT;. This is especially helpful for long SQL based programs that only contain the PROC SQL statement at the beginning and only selected SQL statements should be executed.

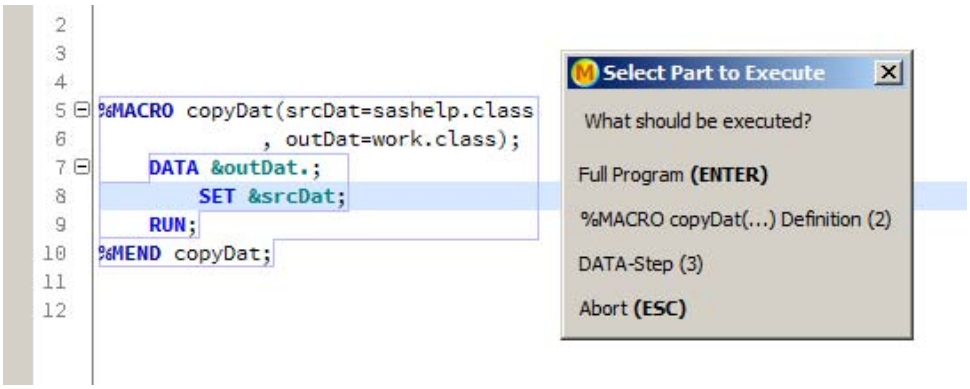

Figure 2: Run Step Dialog Window

"RUN IN TEMPLATE" - EXECUTION WITH CUSTOM TEMPLATE

Sometimes it is helpful to execute additional code around the real code that is to be executed.

Beside the SQL example above, another example is the debugging of a PROC REPORT Step that is used to create an RTF table:

```
ODS RTF ...;
PROC REPORT ...; * Create first table;
RUN;
* Do some other stuff;
PROC REPORT ...; * Create last table;
RUN;
ODS RTF CLOSE;
```

In the program, the PROC REPORT step is just one of many that contribute to the final RTF file. When doing layout optimization, e.g. column widths, a single PROC REPORT step should recreate the RTF file. To achieve this, the program would have to be modified to contain the ODS RTF commands before and after the table.

MACUMBA deals with this issue by defining an ODS RTF template and executes the code surrounded by this template as displayed in Figure 3. This prevents the developer from having to change the code just for development or debugging purpose that have to be undone afterwards. This removes the risk of the developer inadvertently forgetting to remove this temporary code and thus removes the risk of this code affecting the results in the final program.

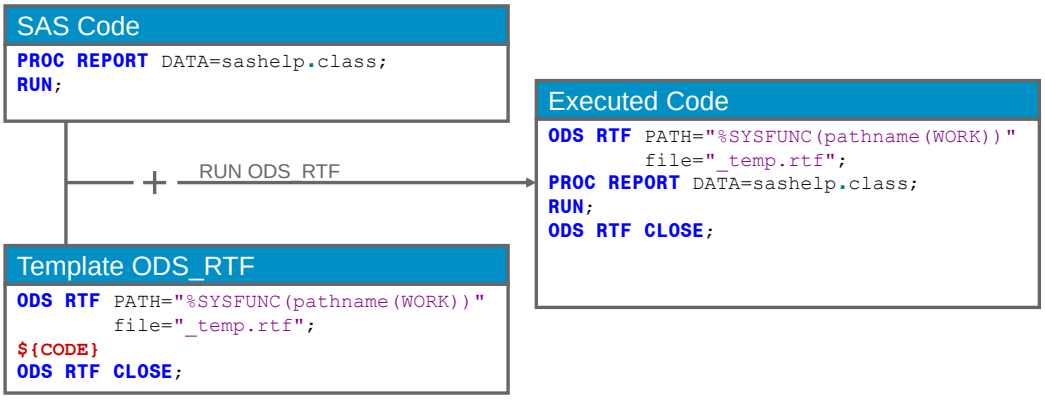

Figure 3: "Run in Template" Example

"RUN AS MACRO" - PARTIAL MACRO EXECUTION

When developing, testing and debugging macros, programmers often comes to the point where a part of the macro should be executed that contains "macro only" code.

For example if from the following macro:

```
%MACRO abc(param1=Y, param2=N);
    %PUT A lot of stuff is to be done;
    %LOCAL i;
    %DO i=1 %TO &max.;
        %IF %SYSFUNC(mod(&i., 2)) EQ 0
        %THEN %DO;
            %PUT &param1. &param2. (i=&i.);
        %END;
    %END;
    %PUT a lot more stuff is to be done;
%MEND;
```

Only the following part is to be executed:

```
%DO i=1 %TO &max.;
    %IF %SYSFUNC(mod(&i., 2)) EQ 0
    %THEN %DO;
        %PUT &param1. &param2. (i=&i.);
    %END;
%END;
```

Than the following error message will be issued:

```
ERROR: The %DO statement is not valid in open code.
```

Additionally the following warning is issued, in case the macro variable param1 does not exist:

```
WARNING: Variable param1 can not be resolved;
```

The common solution would be to modify the macro code accordingly, execute it and undo the modifications afterwards. Again this could cause issues if the modifications are not undone correctly.

MACUMBA provides the execution mode “Run as Macro” that checks the code for surrounding %MACRO definitions and displays a dialog (Figure 4) that allows initializing the macro parameters and required macro variables. Then a temporary macro is created and executed based on the selected macro code and the entered information.

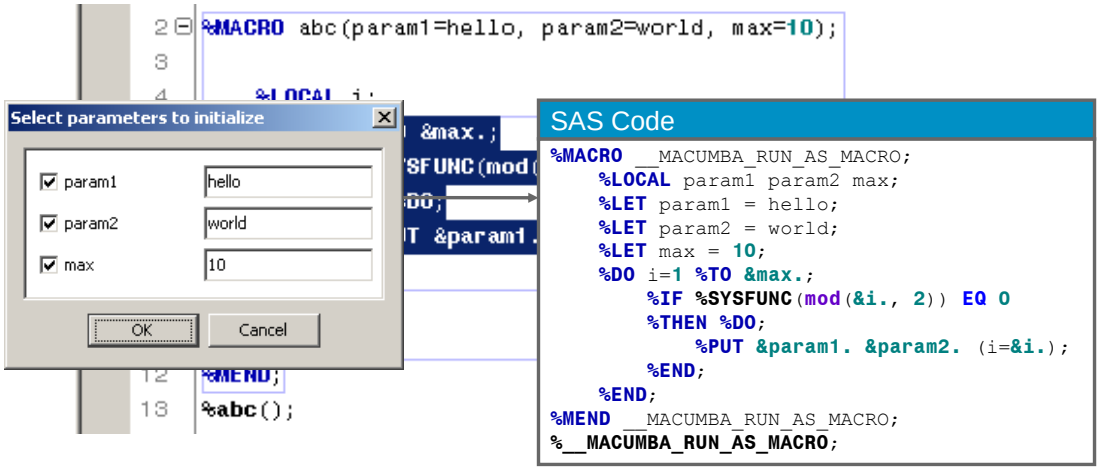

Figure 4: "Run as Macro" overview

“RESOLVE CODE” - A GRAPHICAL MPRINT IMPLEMENTATION

Programmers often use the MPRINT system option when debugging SAS code and often the output in the log is confusing and of little help in finding the issue with the SAS code.

To avoid cutting the MPRINT output line by line from the log, the MFILE option can be used to store it directly in a file. This provides a handy way to debug SAS code that is generated by a macro call.

If for example the SAS code that is generated by the following macro call is to be debugged:

```
%doStudyEvaluation(studyId=0815, projectId=4711);
```

The following code would be executed:

```
FILENAME mprint "%SYSFUNC(pathname(WORK))/resolved.sas";
OPTIONS MPRINT MFILE;
%doStudyEvaluation(studyId=0815, projectId=4711);
```

```
OPTIONS NOMPRINT NOMFILE;
FILENAME mprint;
```

In MACUMBA this could be done by the “Run In Template” feature and a corresponding template. Navigating to the temporary directory and open the file is still a manual task. MACUMBA provides a single step solution in the form of the “Resolve Code” execution mode. When using this, all manual steps are done automatically and additionally a source code formatter reformats the MPRINT output before it is opened.

GRAPHICAL DATA STEP DEBUGGER

One of the well hidden features of PC-SAS is the DATA step debugger. Beside the fact that it is already available since SAS 6.11 and still not known well or used by many SAS programmers, it is a very handy feature when debugging complicated DATA steps.

Among others, the following features are provided by the SAS DATA step debugger:

- Single step / command code execution (GO, JUMP, STEP)
- Breakpoints, when and watch expressions (BREAK, BREAK WHEN, WATCH)
- Examination and updating of variable values (EXAMINE, SET)

PC-SAS IMPLEMENTATION

To start the debugger in PC-SAS the DEBUG parameter has to be added to the DATA command:

```
DATA class / DEBUG;
    DO UNTIL (_last);
        SET sashelp.class END=_last;
        h2  = height**2;
        bmi = (weight / h2) * 703.07;
        bmi = round(bmi, 0.01);
        OUTPUT;
    END;
RUN;
```

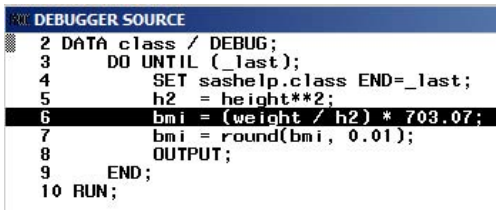

Figure 5: PC-SAS DATA Step Debugger

When executing this code in PC-SAS, the DATA step debugger is started (Figure 5) and waits for debug commands. When this code is executed in a background session without a user interface (e.g. Batch-SAS) the following error message is printed to the log:

```
ERROR: Unable to initialize the DATA STEP Debugger environment.
```

The PC-SAS implementation is strictly command based. Commands are for example STEP, GO, JUMP, EXAMINE, BREAK …. These commands have to be typed each time a command is to be executed. Abbreviations are available, such as E can be used instead of the EXAMINE command, but the code “E <VARIABLE>” has to be typed into the debugger for each variable that should be examined.

MACUMBA IMPLEMENTATION

MACUMBA provides a custom implementation of the DATA Step debugger. Here the debugger is fully integrated into the program editor (Figure 6), this means no extra view is opened, and is completely controlled by mouse actions and keyboard shortcuts.

Some benefits are provided by the MACUMBA integration:

- Code does not have to be changed for debugging
- Breakpoints can be kept over multiple debug sessions (line numbers do not change)
- All features from the editor window (e.g. syntax highlight) are available
- Variable values can be viewed in a separate view and are displayed as a tool tip text, when mouse rests over a variable name
- Jumping in the code can be done mouse based through Drag’n’Drop
- Editing of a long values is more easy (edit instead of retype)

The current implementation does not include all of the features the SAS DATA step debugger provides. For example “BREAK WHEN <*condition*>” is currently not implemented but could be added in a future release.

MACUMBA only “emulates” an interactive session, so the DEBUG parameter can’t be used. For batch sessions, SAS provides the LDEBUG parameter instead:

```
DATA class / LDEBUG;
    DO UNTIL (_last);
        SET sashelp.class END=_last;
        h2  = height**2;
        bmi = (weight / h2) * 703.07;
        bmi = round(bmi, 0.01);
        OUTPUT;
    END;
RUN;
```

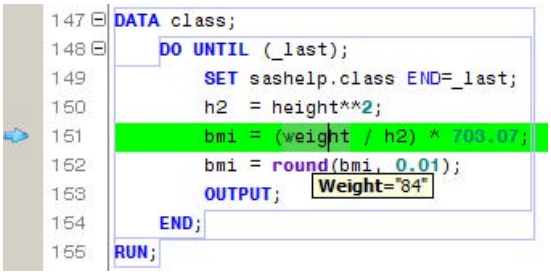

Figure 6: MACUMBA DATA step Debugger

All the rest of the magic is simply the invocation of the correct debugger command at the right time and parsing of the SAS log for debugger output.

GRAPHICAL MACRO DEBUGGER

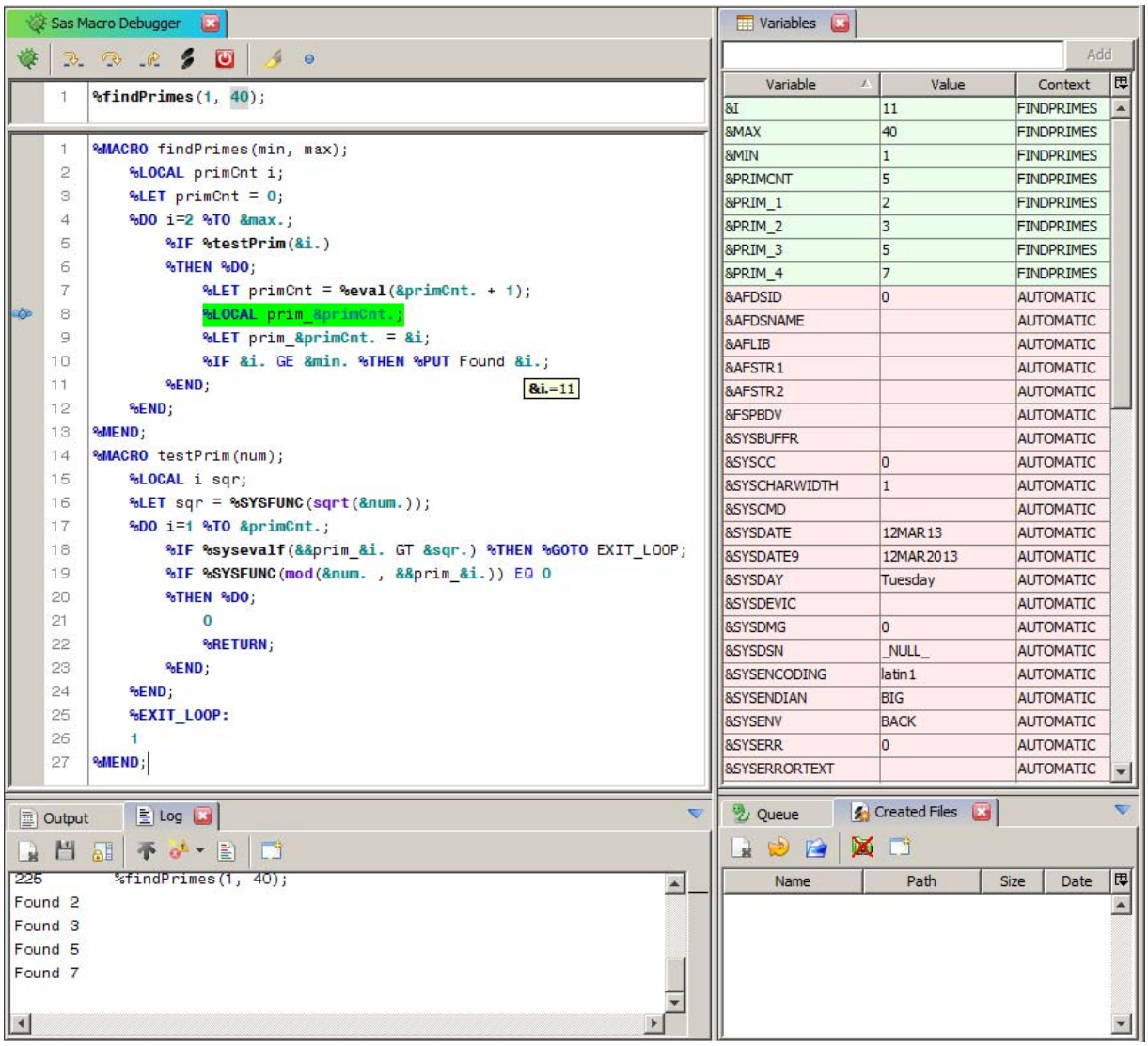

Figure 7: MACUMBA Macro Debugger

A single step debugger for SAS macros is something really missing in a standard SAS development environment. Tracking down a bug in a long and complex macro can become a real challenge.

The two possibilities that are available are to add extra code (e.g. %PUT statements) to strategic positions on the one hand and the usage of SAS system options like SYMBOLGEN or MLOGIC on the other. Both of these provide a way to get the job done, but none of them is really user friendly.

MACUMBA contains a special feature that allows interactive single step debugging in SAS macros.

DIFFERENCES BETWEEN DATA STEP AND MACRO DEBUGGING

When comparing the required debugging functionality of a DATA step with a macro, there are three important differences to consider:

1. A macro consists of a macro definition and a separate macro call. The DATA step only has a definition.
2. A macro can call other macros. A DATA step can only create other DATA steps (e.g. by CALL EXECUTE) but not invoke them within its execution.
3. A macro contains macro code and “normal” SAS code. The “normal” SAS code is considered as text inside of a macro execution. The DATA step only contains “normal” SAS code.

These three differences directly influence the required functionality of the debugger. For example a requirement is that it should be possible to debug multiple macros at the same time. Another requirement is that it has to be possible to supply the invoking macro call separately to the source code of the macros.

SAS MACRO DEBUGGER FEATURES

Based on the general requirements to a debugger and the special requirements for macro debugging, a list of requirements is created that results in the following required features:

- Interrupt code execution
- Perform single step code execution
- Move execution pointer
- Examine and change session state (macro variable values, options, work data sets) when execution is interrupted
- Debug of multiple macros (e.g. one macro is invoking another macro)

Additionally to the required features, the following special features are implemented:

- Execute external code inside of a macro
- Code coverage visualization

SAS MACRO DEBUGGER IMPLEMENTATION

The macro debugger is implemented through the SAS IOM technology. This technology allows accessing a Batch SAS session as it would be an interactive one. That way SAS code can be executed and afterwards the session state (macro variables, options, work data sets and so on) can be evaluated and changed before the next SAS code is executed.

Interrupt Code Execution

For single step debugging, the macro execution is interrupted after each step and the debugger waits for user input. There are six different actions implemented that can be performed by the user, when execution is interrupted:

- Step Over – Execute a single command and wait for the next user input afterwards. If this single step is a call to a macro that is also debugged, the complete sub-macro is executed before the debugger stops again.
- Step Into – Execute a single command and wait for the next user input afterwards. If this single step is a call to a macro that is also debugged, the debugger stops before the first command in the called sub-macro.
- Step Return – Continue the current macro execution. In case the current macro is a sub-macro, the debugger stops again before the next command in the calling macro.
- Go – Continue the code execution.
- Abort – Abort the execution of all debugged macros at the current point. In case the debugged macros are called by a macro that is not debugged, the calling macro is not aborted and will continue.
- Jump – Moves the execution pointer to a different place in the macro.

For all actions (except Abort and Jump) a breakpoint will always interrupt the code execution. So even when “Step Over” is used to not interrupt within a sub-macro, the debugger will interrupt on a breakpoint within this sub-macro.

When considering the special fact, that normal SAS code is considered as text within a macro, a special implementation is needed to handle single stepping within this text. For example the request to be able to interrupt a macro between two DATA steps requires that those two DATA steps are considered as two text blocks within the macro. This might not always be the case by default and the debugger has to consider this when compiling the macro.

Examine and Change Session State

One important feature for debugging is the possibility to get some status information of the current process, when the execution is interrupted. For a macro this contains for example the values of macro variables, options or the content of data sets in WORK or other assigned libraries.

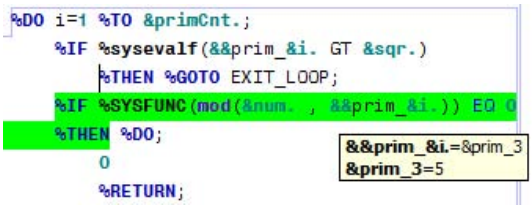

Figure 8: Macro Variable Tool Tip

The SAS IOM technology allows direct access to all this required information, so the focus in development was set on an intuitive display to the user. Special Views are available that list all available macro variables and system options with their values. Another view provides access to Data Sets, Views and Catalogs. Additionally macro variable values are provided as a tool tip text (Figure 8).

Execute External Code Inside of a Macro

As already mentioned above, the SAS IOM technology allows sending SAS code to a SAS session for execution. This feature is used to support code execution even within a suspended SAS macro.

One limitation of this functionality is the restriction of SAS to not allow special macro code outside of a macro. Commands like %IF or %DO have to be compiled by the macro compiler before execution, so they can’t be sent to the SAS connection for direct execution. This restriction is also true for the execution of external code, because this code is not provided at compile time, it is not allowed to contain those macro statements.

Code Coverage Visualization

The code coverage visualization is a special feature used for validation purpose. It is not related to debugging, but the implementation is almost the same.

One of the main questions in validation is about how many tests have to be done to cover all possible scenarios. Code coverage visualization does not answer this question completely, but it allows finding parts in the code, that are not executed by any of the tests. With this information additional tests can be developed that consider these lines.

Within the debugger, code coverage can be implemented by storing all executed commands in a list. At the end this list can be evaluated to find commands not executed. Additionally it is possible to count how often a command was executed.

The current implementation only measures what code is executed in a macro, therefore conditions in “normal” SAS code are not evaluated. This means as soon as a DATA step is compiled through a macro, all code in the DATA step is marked as executed, even when special IF conditions never evaluate to true at runtime.

SAS MACRO DEBUGGER USAGE

The SAS macro debugger is implemented as a special view (Figure 7) that provides two text fields. One field contains the macro call and the other contains the macro definition. In case the macro is accessible through the “autocall” macro facility, only the macro call has to be given, the macro code can be fetched automatically. Otherwise the macro code can be copied manually into the code field. For already compiled macros, the source code can be retrieved from the compiled macro (decompiled), but this would not include macro or block comments (removed on compilation) and is therefore not recommended.

The rest of the usage is similar to the DATA step debugger. Debugging can be started through a toolbar button or a keyboard shortcut. The same is true for the GO and the STEP actions. Breakpoints can be added and removed by mouse clicks and the instruction pointer can be moved with the mouse as well. Values of macro variables can be seen in the “Variables” window or as a mouse over “tool tip text”.

In case the macro can’t be invoked on a “standalone” basis, the code in the macro call text field can be adapted. This allows, for example, to invoke an inline macro in a %PUT statement or in a DATA step.

One other example where this separation between call and definition is very helpful is the possibility to debug macro “A” when invoking macro “B” where macro “A” is only called internally by macro “B”.

Additionally it is possible to debug multiple macros at the same time. Therefore the macro code of all macros is to be copied into the macro code field. Within macro debugging breakpoints can be set to any of the macros and in single step mode a macro call will enter debugging of the called macro.

FEATURE OUTLOOK

Additionally to the currently implemented features, some more will come. These include:

- Support for statement evaluation (Watch Expressions) when suspended
- Support full macro code modification within debugging
- Automatically add sub macros to be debugged on invocation if requested
- Support for the DATA step debugger within a debugged macro
- Single step debugging for special procedures (FCMP, REPORT, DS2, …)
- Correct Code Coverage Visualization for “normal” SAS code.

CONCLUSION

The SAS IOM technology is a very powerful feature of SAS. Through SAS IOM it was possible to create a SAS development environment that perfectly fits our needs and provides state of the art features for SAS program development and debugging.

The implementation of the single step macro debugger was a special challenge that would not have been possible without the well documented SAS API.

REFERENCES

- http://support.sas.com last access October 9, 2012.
- S. David Riba “How to Use the Data Step Debugger” S. David Riba, JADE Tech, Inc., Clearwater, FL http://www2.sas.com/proceedings/sugi25/25/btu/25p052.pdf (SUGI paper 52-25) last access October 9, 2012.
- SAS “SAS® 9.2 Integration Technologies: Java Client Developer’s Guide” Copyright © 2009, SAS Institute Inc., Cary, NC, USA, ISBN 978-1-59994-846-1 http://support.sas.com/documentation/cdl/en/itechjcdg/61499/PDF/default/itechjcdg.pdf last access October 9, 2012.

ACKNOWLEDGMENTS

All this would not have been possible without the support of Elena Glathe. Your challenging requests push me every day a little further.

RECOMMENDED READING

- SAS® 9.2 Integration Technologies Overview
- SAS® 9.2 Integration Technologies Java Client Developer’s Guide

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Michael Weiss
Enterprise: Bayer Pharma AG
Address: Muellerstr. 178, P300
City, State ZIP: Berlin, 13353 , Germany
Work Phone: +493046817687
E-mail: michael.weiss@bayer.com