



# Effective Scripting in JMP

HMS Analytical Software GmbH - Dr. P. Warnat

PhUSE 2009



# Background: What is JMP?



- application for statistical analysis
- interactive and easy-to-handle user interface

---

## Comparison JMP and SAS

JMP



SAS



JMP & scripting

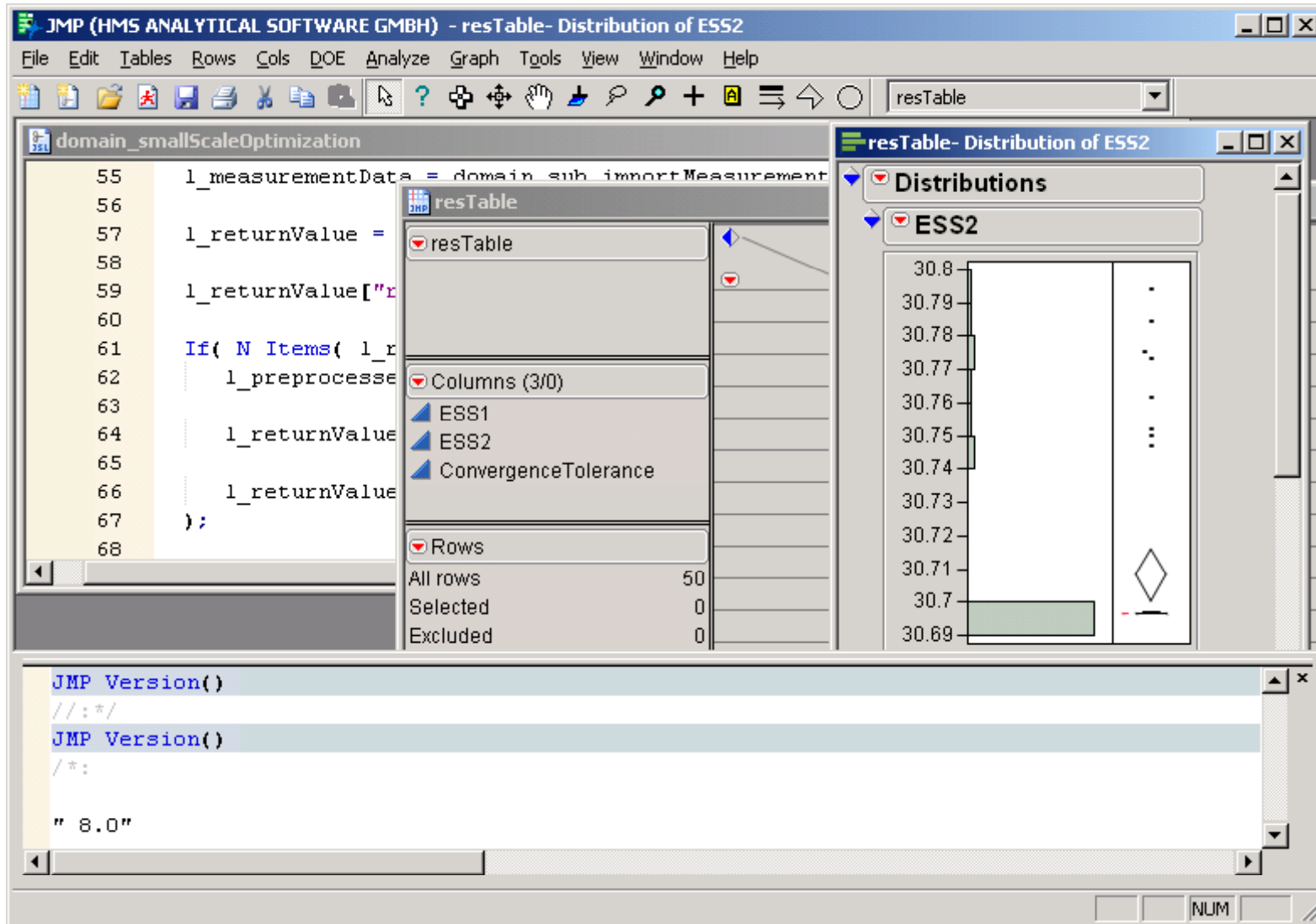


# Overview

- **Introduction JSL scripting**
- **How to work effectively with JSL?**  
Recommendations regarding the topics:
  - STRUCTURING AND REUSE
  - UNIT-TESTS
  - DEPLOYMENT
  - CUSTOM REPORTS
  - EXTERNAL TOOLS
- **Conclusion**



# Introduction JSL scripting



The screenshot displays the JMP software interface with the following components:

- Script Window:** Contains JSL code for data import and processing:

```
55  l_measurementData = domain_sub import Measurement
56
57  l_returnValue =
58
59  l_returnValue["r
60
61  If( N Items( l_r
62    l_preprocesse
63
64    l_returnValue
65
66    l_returnValue
67  );
68
```
- Table Window (resTable):** Shows a table with 3 columns: ESS1, ESS2, and ConvergenceTolerance. The 'Rows' section indicates 50 total rows, with 0 selected and 0 excluded.
- Distributions Window (ESS2):** Displays a histogram for the ESS2 variable. The y-axis ranges from 30.69 to 30.8. A single bar is visible at the bottom left, and a diamond-shaped outlier marker is present on the right side of the plot.
- Console Window:** Shows the output of the `JMP Version()` command, which is " 8.0".

# Introduction JSL scripting

- **Jmp Scripting Language (JSL):**
  - automation of a sequence of tasks within JMP
  - implementation of your own algorithms within JMP
  - syntax: “functional” programming approach
  - JMP environment components represented as objects with attributes and methods
  - JSL is well documented in the JMP scripting guide

# How to work effectively with JSL?



**The following slides render a personal list of useful tips resulting out of lessons learned in several mid-size projects utilizing JSL scripting**



# STRUCTURING AND REUSE

- **Define own functions**
- **Use local variables**
- **Use local error handling in every function**
- **Structure your functions in "modules" with naming conventions**
- **Use include to use function definitions out of other files**



# Example – function template

```
<FUNCTION NAME> = FUNCTION( {<ARGUMENTS>},  
    LOCAL( {<LOCAL VARIABLES>},  
    TRY(  
        <FUNCTION BODY SCRIPT CODE>  
    , //CATCH  
        <ERROR HANDLING SCRIPT CODE>  
    );//END TRY  
    );//END LOCAL  
);//END FUNCTION
```

# Example – real function

```

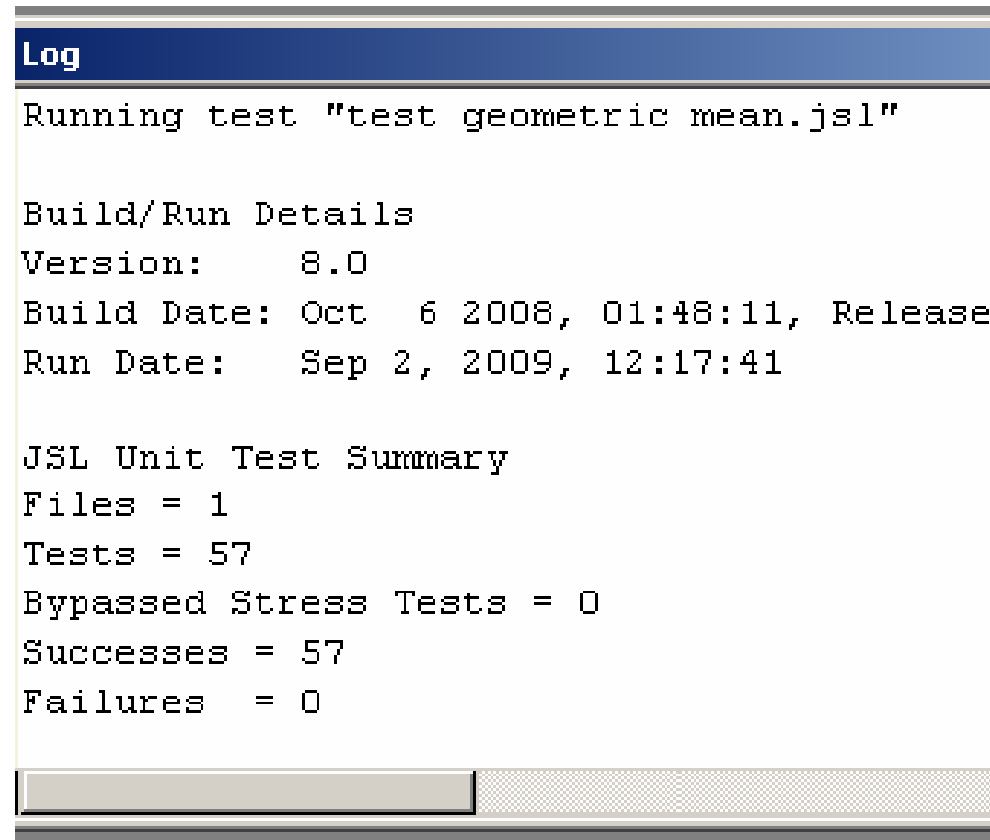
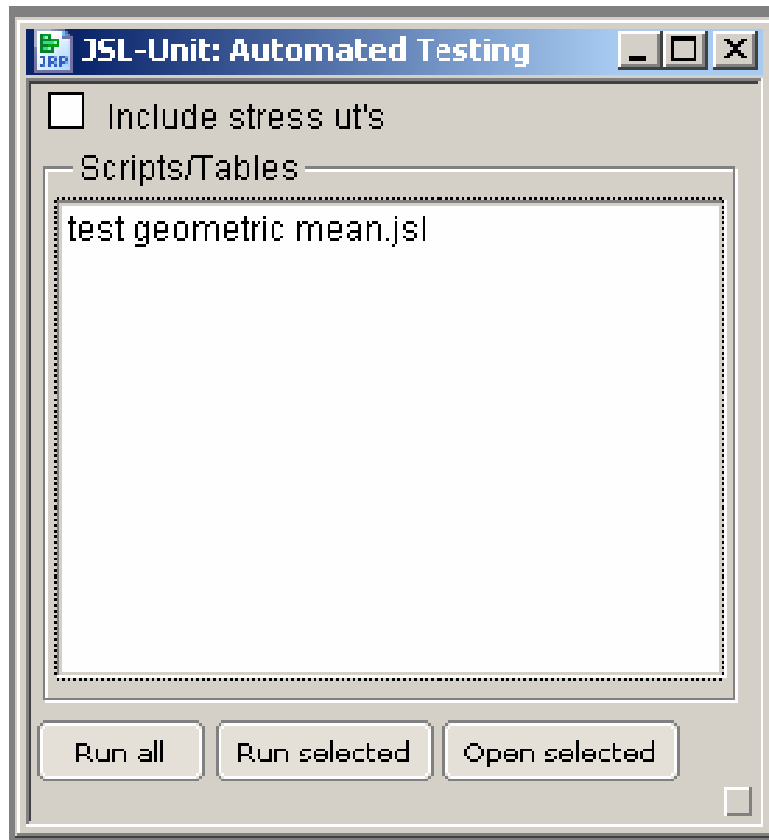
util_getSumOfListElements = Function( {i_List},
  Local( {l_index, l_return},
    l_return = 0;
    Try(
      For( l_index = 1, l_index <= N Items( i_List ), l_index++,
        l_return = l_return + i_List[l_index]
      );
    ,
      Throw( "Unexpected ERROR in util_getSumOfListElements: " ||
        Char( exception_msg )
      );
    ); //End Try
  l_return;
); // End Local
); // End Function

```

# UNIT-TESTS

- **To get JSL code that runs reliable -> run tests!**
- **For JSL, a unit-test framework named JSL-Unit is available (JMP website)**
- **Implement your tests as JSL code**  
**Advantages:**
  - Executable documentation of what you have tested
  - enables you to repeat execution of your tests easily (to check whether later modifications on your functions have unintended side effects)

# JSL Unit Example



# DEPLOYMENT

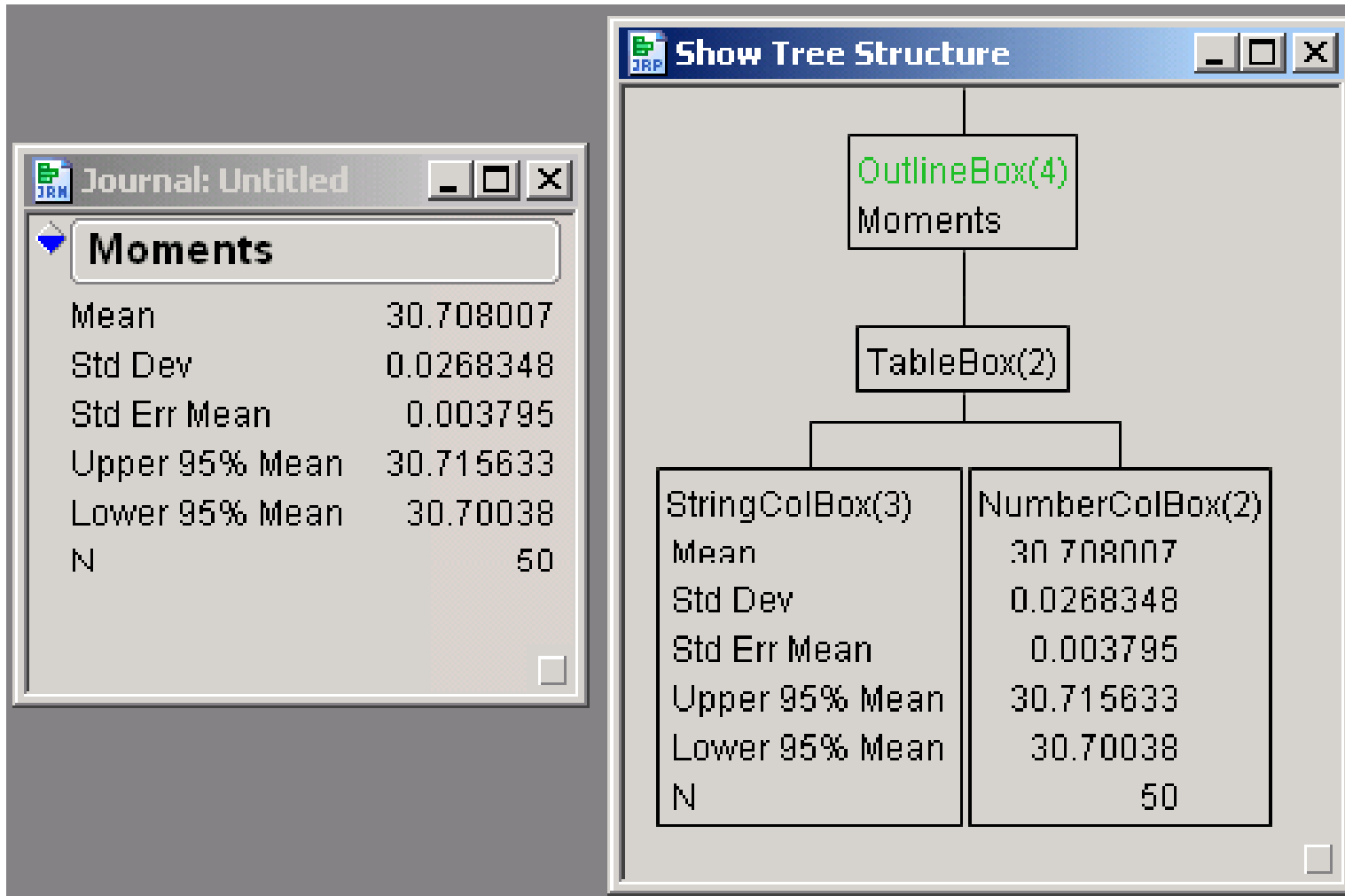
- **Provide JSL code to colleagues or customers:**
  - Use “//!” for automatic execution on opening the script file
  - Hide JSL code with script encryption
  - Extend JMP menus with new scripts or use
  - “deployment containers” for scripts, like JMP journal and project files.  
Both of them can contain scripts or links to script files

# CUSTOM REPORTS

- **Reuse components of reports generated by build-in JMP methods**
- **The display tree visualization of an existing report helps a lot**
- **To get the display tree for a given report window hold down the Control and Shift keys while right-clicking on a blue triangle in the report; select Edit > Show Tree Structure**



# Example – Display Tree



The screenshot displays two windows from the HMS software. The left window, titled 'Journal: Untitled', shows a table of statistical moments. The right window, titled 'Show Tree Structure', displays a hierarchical tree diagram of the data structure.

**Journal: Untitled - Moments**

Mean	30.708007
Std Dev	0.0268348
Std Err Mean	0.003795
Upper 95% Mean	30.715633
Lower 95% Mean	30.70038
N	50

**Show Tree Structure**

```
graph TD; A[OutlineBox(4)  
Moments] --> B[TableBox(2)]; B --> C[StringColBox(3)  
Mean  
Std Dev  
Std Err Mean  
Upper 95% Mean  
Lower 95% Mean  
N]; B --> D[NumberColBox(2)  
30.708007  
0.0268348  
0.003795  
30.715633  
30.70038  
50];
```

# EXTERNAL TOOLS

- **Use external software tools to support your JSL code development**
  - **a version control system**  
(e.g. Subversion; <http://subversion.tigris.org/>)  
→ recover old versions of your code whenever you want or need
  - **a source code documentation system**  
(e.g. Doxygen; <http://www.doxygen.org>)  
→ generate API documentation out of source code documentation
  - **a build tool**  
(e.g. Apache Ant; <http://ant.apache.org/>)  
→ automate some tasks  
  
(e.g. copy actual version of your source code files and archive them into a zip-file)

# Conclusion

- **JMP: nice and easy-to-handle user interface**
- **JSL scripting: a mighty scripting language for automation or enhancement of JMP functionality**
- **The recommendations presented here will help you implementing even large projects effectively with JSL**
- **Give it a try!**



# Thank you for your attention



**Dr. Patrick René Warnat**

**HMS Analytical Software GmbH  
Rohrbacher Str. 26  
69115 Heidelberg  
Germany**

**[www.analytical-software.de](http://www.analytical-software.de)**

