```
lt IS %_SWITCHED_ON.
>SCONT time 0 0. "GET TIM
: / 'Global Trade Managem


1. "=> Makes Code Inspect
FORM run.
RM out.
```

ABAP System Deep Dive

Das SAP-System ist uralt und steckt voller Geheimnisse. Einige alte Artefakte sind immer noch vorhanden. Außerdem gibt es einige versteckte Kommandos. In diesem Artikel möchte ich dir einiges von dem *geheimen Wissen* vorstellen.

```
1   REPORT.
2
3   IF ea-glt IS %_SWITCHED_ON.
4     RSYN >SCONT time 0 0. "GET TIME
5     WRITE: / 'Global Trade Management aktiv'.
6   ENDIF.
7
8   IF 1 I 1. "=> Makes Code Inspector dump
9   #r3 PERFORM run.
10    PERFORM out.
11  ENDIF.
12
```

1. R/2, R/3-Präprozessor

Es besteht die Möglichkeit, bestimmte Anweisungen nur auf R/2- bzw. R/3-Systemen auszuführen. Dies geht mithilfe des Präprozessor-Befehls #rx, wobei x für das entsprechende System steht.

R/2-Systeme interpretieren die Zeichenfolge #r3 am Anfang der Zeile als Kommentar, während R/3-Systeme die Zeilen ignorieren, die mit #r2 anfangen. Dies wird z.B. im Pretty-Printer ausgenutzt.

#r2 WRITE ‚Ich bin ein R/2-System‘.
#r3 WRITE ‚Ich bin ein R/3-System‘.

2. Operator IS %_SWITCHED_ON

Mit diesem versteckten Schlüsselwort kann man prüfen, ob ein bestimmter Switch im Switch Framework gesetzt ist. So kann man etwa ein Verhalten implementieren, das sich zwischen Industrie- und Retail-Systemen unterscheidet. Man kann den Operator u.A. in den Anweisungen CHECK oder IF verwenden.

3. Laufzeitmessung mit +REP/+ENDREP

Es besteht die Möglichkeit, eine Laufzeitmessung in ABAP durchzuführen. Dies funktioniert mit einer speziellen Schleife, der +REP/+ENDREP-Schleife. Die Anweisung hat noch einige Zusatzparameter, z.B. dass man nur zu einem bestimmten Befehl messen oder weitere interne

Statisiken mit auslesen möchte. Das möchte ich hier nicht im einzelnen wiedergeben. Falls Sie Fragen hierzu haben, kann ich hierzu weitere Informationen liefern.

```
DATA ls_rep_results TYPE rep_s_results.

+REP 100 times.
 "hier das Statement einfügen (z. B. Methodenaufruf)
 +ENDREP RESULTS ls_rep_results.

WRITE: (30) 'Brutto-Laufzeit', ls_rep_results-rtime,
 /(30) 'Datenbankzeit', ls_rep_results-dbtime,
 /(30) 'Anzahl der Datenbankaufrufe', ls_rep_results-dbcount,
 /(30) 'Netzwerkzeit', ls_rep_results-ictime,
 /(30) 'Anzahl der Netzwerkzugriffe', ls_rep_results-iccount.

CLEAR ls_rep_results. "Clear nicht vergessen!!!
```

Scheinbar gibt es noch die Möglichkeit, die Performance von einzelnen ABAP-Opcodes (zu sehen im Debugger mit Rechtsklick ABAP-Bytecode) zu messen. Die Syntax ist:

```
  DATA lv_opcode TYPE char4.
  DATA lv_flags  TYPE x.

  lv_opcode = 'WRIB'.
  lv_flags  = '01'.
  BREAK-POINT.
  +REP 1 TIMES UP TO lv_opcode lv_flags.
  WRITE 'foo'.
  +ENDREP RESULTS gs_rep_results.

  lv_opcode = 'TIME'.
  lv_flags  = '00'.
  +REP 100 TIMES REPEATING lv_opcode lv_flags.
  WRITE 'bar'.
  GET TIME.
  WRITE 'baz'.
  +ENDREP RESULTS gs_rep_results.
```

Allerdings ist dies sehr technisch und vor allen Dingen undokumentiert und führt zu Fehlern wie CX_SY_REP_PARAMETER_ERROR und CX_SY_REP_INTERNAL_ERROR.

Man kann außerdem

```
+REP 100 TIMES WITH KERNELINFO.
```

benutzen, dann stehen in den Feldern gs_rep_results-bboxinfo, gs_rep_results-kinfo und gs_rep_results-inlayinfo zudem weitere Infos.

Es handelt sich offenbar um Informationen zu Kernel-Funktionen wie vMemcpyR (Kopieren von Speicherbereichen).

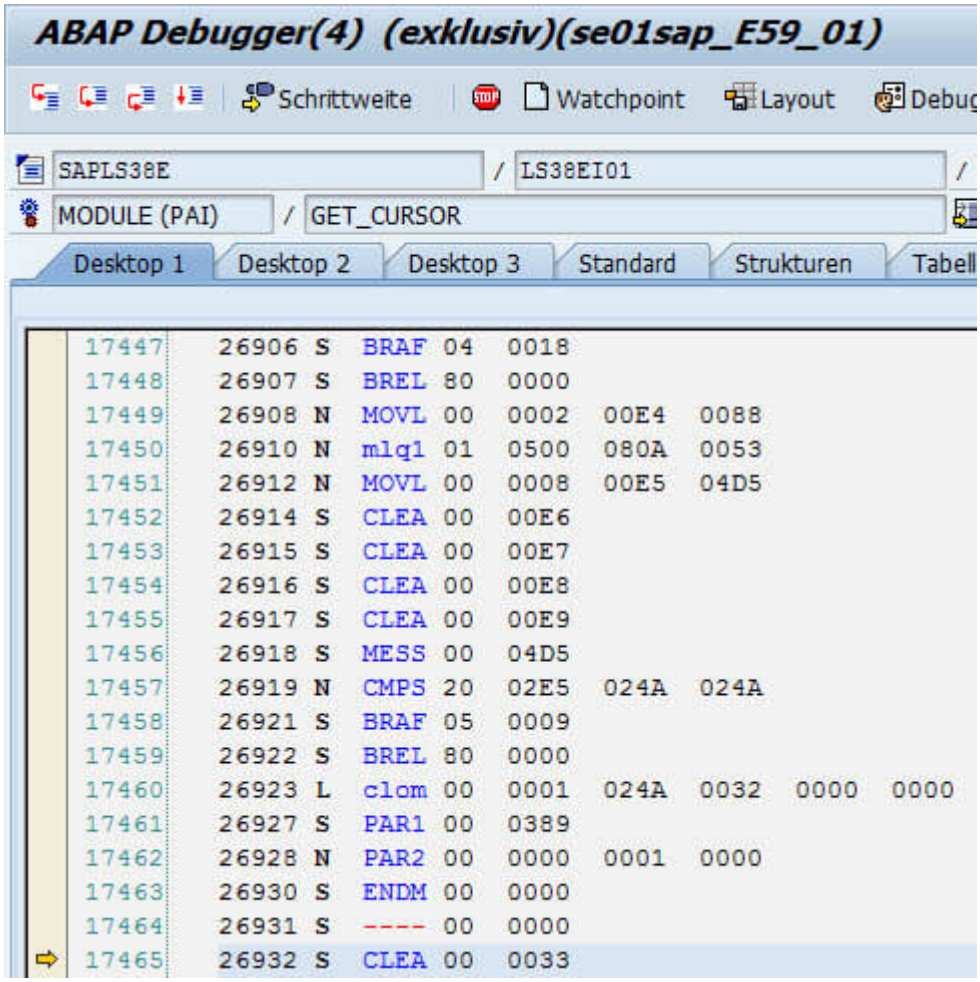
4. Globale Variable in Form-Routine oder Methode einer lokalen Klasse definieren

Es ist möglich, eine lokale Variable innerhalb einer Form-Routinen oder in der Methode einer lokalen Klasse als global zu definieren. Fragt nicht, wozu das sinnvoll ist, aber die Anweisungen DATA unterstützt den Zusatz %_NON-LOCAL, der die Variable global deklariert.

5. Internen Modus bei SUBMIT explizit angeben.

Man kann mit dem Zusatz %_INTERNAL_%_SUBMODE_% bei der Anweisung SUBMIT explizit angeben, in welchem internen Modus ein Programm ausgeführt werden soll. Hiermit kann Interprozesskommunikation implementieren. Die Anweisung wird z.B. im Systemprogramm RDSBRUNT (dient der Steuerung von Reports) verwendet.

6. ABAP-Assembler



Sicherlich bist du schon einmal im Debugger oder in der ST22 über den ABAP-Bytecode gestolpert. Es besteht die Möglichkeit, Bytecodebefehle direkt in ABAP einzugeben. Die interne Anweisung RSYN ermöglicht es mit dem Zusatz >SCONT beliebige Bytecode-Befehle auszuführen. Das erste Argument entspricht dem Namen des Bytecodebefehls, das zweite Argument ist der erste Parameter (binär) und das zweite Argument der zweite Parameter (dezimal).

```
RSYN >SCONT time 0 0.
```

ist gleichbedeutend mit dem ABAP-Statement

```
GET TIME.
```

RSYN >SCONT wird im Standard in der Komponente zum Aufruf von RFC-Calls verwendet, um Spezialformen der Anweisung SYSTEM-CALL FREE MODE aufzurufen. Welche weiteren Funktionalitäten das Schlüsselwort RSYN bietet, konnte ich noch nicht herausfinden.

7. zusätzliche Vergleichsoperatoren

Aufgrund eines Fehlers im Compiler wurden vor ABAP 7.40 beliebige Zeichen als Vergleichsoperatoren erkannt. Dies ist mittlerweile gefixt, aber aus Kompatibilitätsgründen wird der Operator I (der gleichbedeutend mit EQ ist) weiterhin unterstützt.

Folgende Anweisung ist für den ABAP-Compiler eine gültige Syntax:

```
CHECK 1 i 1.
```

8. Vorsicht!

All die hier erwähnten Befehle oder Zusätze sollten nie in einem produktiven SAP-System eingesetzt werden. Die Behandlung dieser Befehle dient nur dem besseren Verständnis des SAP-Systems und um beim nächsten SAP-Stammtisch ein bisschen angeben zu können.

Keiner der hier vorgestellten Befehle ist dem Code Inspector auch nur ein Hinweis wert. Die Befehle könnten also weitestgehend unbemerkt eingesetzt werden.

Der Befehl CHECK 1 i 1 bringt die erweiterte Programmprüfung – die auch im Code Inspector aufgerufen wird – dazu, mit einem Kurzdump abzustürzen.

9. Beispielcode

Ein paar der oben genannten Beispiele sind in folgendem Listing untergebracht:

```
REPORT.

IF ea-glt IS %_SWITCHED_ON.
 RSYN >SCONT time 0 0. "GET TIME
 WRITE: / 'Global Trade Management aktiv'.
ENDIF.

IF 1 I 1. "=> Makes Code Inspector dump
#r3 PERFORM run.
 PERFORM out.
ENDIF.

*&---------------------------------------------------------------------*
*& Form run!
*&---------------------------------------------------------------------*
FORM run.

DATA ls_rep_results TYPE rep_s_results %_NON-LOCAL. "=>GLOBALISIEREN

 +REP 10 TIMES.
 WRITE: / sy-uzeit.
 SELECT COUNT( * ) FROM t001.
 +ENDREP RESULTS ls_rep_results.

ENDFORM. "run!

*&---------------------------------------------------------------------*
*& Form out
*&---------------------------------------------------------------------*
FORM out.

  WRITE: (30) 'Brutto-Laufzeit', ls_rep_results-rtime EXPONENT 0 DECIMALS 0,
        /(30) 'Datenbankzeit', ls_rep_results-dbtime EXPONENT 0 DECIMALS 0,
        /(30) 'Anzahl der Datenbankaufrufe', ls_rep_results-dbcount EXPONENT
0 DECIMALS 0,
```

```
        /(30) 'Netzwerkzeit', ls_rep_results-ictime EXPONENT 0 DECIMALS 0,
        /(30) 'Anzahl der Netzwerkzugriffe', ls_rep_results-iccount EXPONENT
0 DECIMALS 0.

ENDFORM. "out
```