Einführung in z/OS:
2. COBOL mit TSO und ISPF

Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Abteilung Technische Informatik
Prof. Dr. Wolfgang Rosenstiel
Gerald Kreißig, Dr. Jens Müller

7. November 2016

Inhaltsverzeichnis

1 Vorgehen

COBOL ist auf Mainframes immer noch die meist verbreitete Sprache!

In der folgenden Anleitung wird erklärt, wie man ein „Hello World“-Programm in COBOL[1] schreibt, übersetzt und startet. Dies soll unter TSO und ISPF geschehen.

1.1 Einrichten der Entwicklungsumgebung

Um COBOL-Code in ein auführbares Programm zu übersetzen, muss für jedes Zwischenergebnis ein Dataset angelegt werden, dass die Teilergebnisse aufnimmt. In Abbildung 1 ist das Vorgehen beim Übersetzen zu sehen. Unter z/OS werden Programme generell durch Jobs ausgeführt. Diese Jobs lassen sich mithilfe von JCL[2] erzeugen und starten. Für das Kompilieren und Linken wird demnach jeweils ein Job benötigt, der dies durchführt.

Hier sei darauf hingewiesen, dass der mittlere Qualifier nur beispielhaft gewählt ist.

Für die Aufgaben werden folgende Namenskonventionen für die Datasets verwendet:

- `PRAKxxx.TEST.COB` für den Cobol Sourcecode
- `PRAKxxx.TEST.LOAD` für den Object Code des übersetzten Sourcecodes
- `PRAKxxx.TEST.CNTL` für die JCL-Scripte

[1] **CO**mmon **B**usiness **O**riented **L**anguage
[2] **J**ob **C**ontrol **L**anguage
[3] **L**anguage **E**nvironment: von der Programmiersprache bereitgestellte Funktionen, die bereits übersetzt wurden (mathematische Funktionen, Memory Management, etc.)

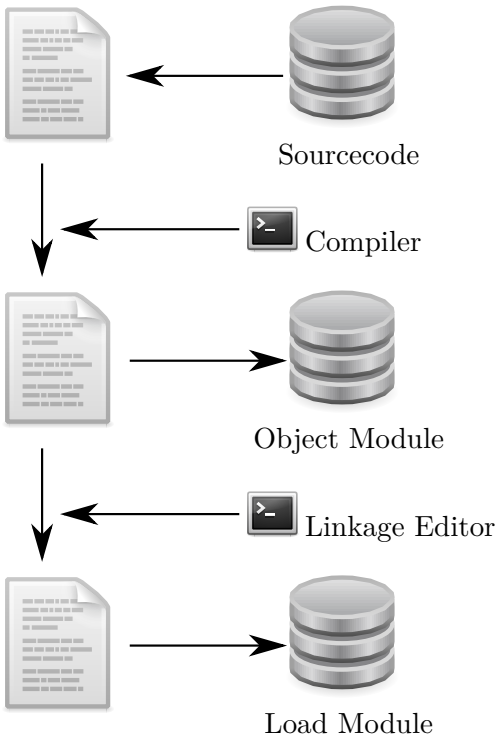
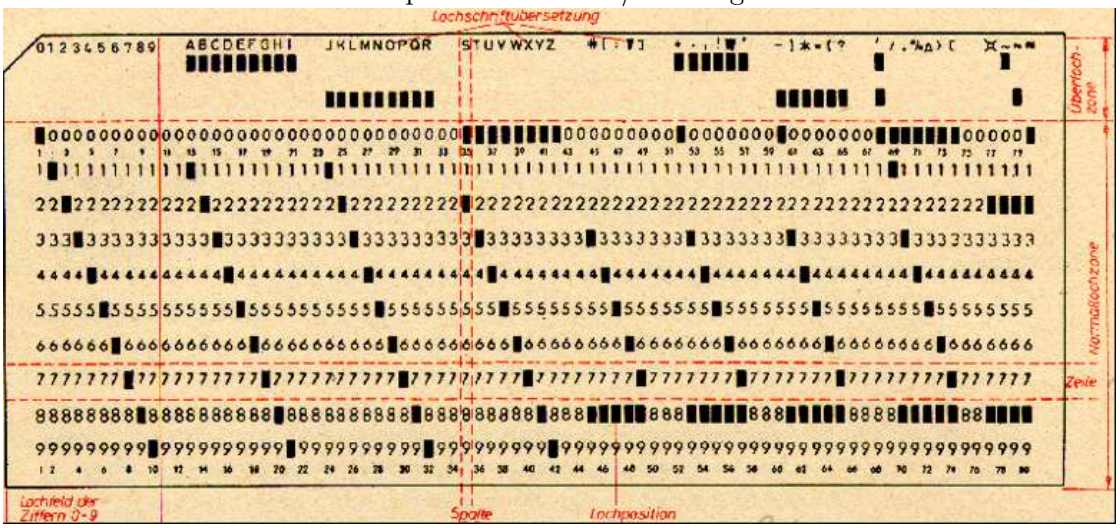


Abbildung 1: **Übersetzen eines Programmes:** Der Sourcecode befindet sich als Member in einem Dataset. Ein PDSe Dataset stellt in vielen Fällen eine Programm-Library dar, wobei in den Membern die einzelnen Programme (oder Programmteile) gespeichert werden. Diese werden jeweils durch den Compiler in Objektmodule übersetzt, die bereits aus System z Maschinenbefehlen bestehen. Da ein Programm allerdings üblicherweise aus mehreren Teil- oder Unterprogrammen besteht, müssen diese in einem weiteren Schritt zu einem einzigen ausführbaren Programm zusammengesetzt werden. Der Linkage Editor löst die symbolischen Referenzen der Objektmodule untereinander auf und ersetzt diese durch Hauptspeicheradressen. Ebenfalls werden von ihm Systemroutinen (Dateizugriff, LE^3) eingebunden. Das resultierende Loadmodul lässt sich letztendlich in den Hauptspeicher laden und ausführen. Dieses Vorgehen bezieht sich auf alle unterstützen Programmiersprachen, wie etwa COBOL, PL/1 und C/C++.

1.2 Erstellen des COBOL-Sourcecodes

Wenn man COBOL-Code über den ISPF-Editor erstellt, ist man mit dem Problem der Spaltenabhängigkeit konfrontiert. Diese geht noch aus Zeiten der Lochkarte hervor und ist auf Grund der Aufwärtskompatibilität aller z/OS-Programme immer noch erhalten.



Die Entwicklung der maschinellen Informationsverarbeitung begann mit der Erfindung der Lochkarte durch Herrn Herrmann Hollerith, dem Gründer der Firma IBM. Abgebildet ist eine Lochkarte der Maße 19 x 8 cm, wie sie 1928 von IBM eingeführt und standardisiert wurde, und bis etwa 1970 in Gebrauch war. Der Vorläufer mit einem etwas anderen Format wurde von Herrmann Hollerith 1890 bei der Volkszählung in den USA und wenig später auch in Deutschland eingesetzt.

Eine Lochkarte kann eine Zeile à 80 Spalten (bzw. Zeichen) aufnehmen. Dies entspricht also einer Zeile im ISPF-Editor. Sollte ein Statement länger als 80 Zeichen sein, musste an einer bestimmten Stelle eine Stanzung stattfinden (analog Übertrag) und die Vortsetzung auf einer weiteren Karte erfolgen. Im ISPF-Editor wird dies durch ein Zeichen an 7-er Spalte einer Zeile signalisiert.

Für COBOL bestehen die Konventionen, dass `Labels` in **Spalte 8** und Befehle in **Spalte 12** beginnen müssen. Eine Übersicht gibt:

Spalte	Zweck
1-6	Nummerierung
7	Zeilenkennzeichen: „-“ für Folgezeile, „*“ und „/“ für Kommentare, „+“ für Seitenwechsel
8-11	Bereich A (Label/Überschriften)
12-72	Bereich B (Befehle)
73-80	frei

Diese Spaltenabhängigkeit wird vom COBOL-Compiler für z/OS erwartet und bei Nichteinhaltung gibt es entsprechende Fehlermeldungen in den Logs. Die von Mainframes verwendete Codepage EBCDIC[4] entspricht genau der 8-bit Umsetzung der BCD-Lochkartenstanzungen.

[4] **E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode (Kurzaussprache: „Eb-Stick“ :-))

Ein COBOL-Programm ist in Teile (`DIVISION`), Kapitel (`SECTION`) und Abschnitte (`PARAGRAPH`) gegliedert. Die vier `DIVISION`s sind in ihrer festgelegten Reihenfolge:

- `IDENTIFICATION DIVISION`: Programmnamen und Kommentare
- `ENVIRONMENT DIVISION`: Definition von Schnittstellen zum Betriebs- und Dateisystem
- `DATA DIVISION`: Definition der Programmvariablen und Datenstrukturen
- `PROCEDURE DIVISION`: eigentlicher Code der prozeduralen Anweisungen

ENVIRONMENT und DATA DIVISIONs können unter Umständen ganz entfallen.

Es ist hieraus ersichtlich, dass eine strikte Trennung von Datendeklaration und prozeduralen Anweisungen erfolgt, durch die sich COBOL auszeichnet. In der Procedure Division kann man nur Variablen benutzen, die vorher in der Data Division deklariert worden sind.

Der Cobol-Code für das „Hallo Welt“-Programm ist in einen Member des `x.x.COB`-Datasets zu schreiben und sieht folgendermaßen aus:

`x.x.` sei hier als Platzhalter zu verstehen für die in Abschnitt 1.1 verwendeten Dataset-Qualifier.

```
␣␣␣␣␣␣␣␣IDENTIFICATION␣DIVISION.
␣␣␣␣␣␣␣␣PROGRAM-ID.␣COB01.
␣␣␣␣␣␣␣␣ENVIRONMENT␣DIVISION.
␣␣␣␣␣␣␣␣INPUT-OUTPUT␣SECTION.

␣␣␣␣␣␣␣␣FILE-CONTROL.
␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣PRINTOUT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ASSIGN␣TO␣SYSPRINT.
␣␣␣␣␣␣␣␣DATA␣DIVISION.
␣␣␣␣␣␣␣␣FILE␣SECTION.
␣␣␣␣␣␣␣␣FD␣␣␣␣␣␣PRINTOUT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣RECORD␣CONTAINS␣80␣CHARACTERS
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣RECORDING␣MODE␣F
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣BLOCK␣CONTAINS␣0␣RECORDS
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣LABEL␣RECORDS␣ARE␣OMITTED.
␣␣␣␣␣␣␣␣01␣␣PRINTREC␣␣␣␣␣␣␣␣␣PIC␣X(80).
␣␣␣␣␣␣␣␣WORKING-STORAGE␣SECTION.

␣␣␣␣␣␣␣␣LINKAGE␣SECTION.
␣␣␣␣␣␣␣␣PROCEDURE␣DIVISION.
␣␣␣␣␣␣␣␣anfang.
␣␣␣␣␣␣␣␣␣␣␣␣OPEN␣OUTPUT␣PRINTOUT.
␣␣␣␣␣␣␣␣␣␣␣␣move␣'Hallo␣Welt,␣unser␣erstes␣TSO-PROGRAMM␣in␣COBOL'␣to␣prin
␣␣␣␣␣␣␣-␣␣␣␣trec.
␣␣␣␣␣␣␣␣␣␣␣␣WRITE␣PRINTREC.
␣␣␣␣␣␣␣␣␣␣␣␣CLOSE␣PRINTOUT.
␣␣␣␣␣␣␣␣␣␣␣␣GOBACK.
```

Minuszeichen beachten, der Name der Variablen ist „printrec“

1.3 Erstellen der JCL-Scripte

Wenn ein Quellprogramms unter TSO/ISPF entwickelt wird, erfolgt die Übersetzung typischerweise mittels eines JES[5]-Stapelberarbeitungsprozesses. Der Auftrag hierfür wird mittels eines JCL-Scriptes erstellt, das Linken mehrer Object Module zu einem Load Modul durch den Linkage Editor erfordert ein weiteres JCL Script. Wie bereits erwähnt, existiert bei JCL ebenfalls eine Spaltenabhängigkeit. Ein JCL Script besteht aus einzelnen JCL Statements. Ein Statement passte früher auf eine Lochkarte mit 80 Spalten und daran hat sich bis heute nichts geändert. Ein Statement besteht aus 5 Feldern:

//	Name	Operation	Parameter	Kommentar

Ein Statement beginnt immer mit „//“, woraufhin das Namensfeld in Spalte 3 folgt mit einer maximalen Länge von 8 Zeichen. Die weiteren Felder folgen jeweils durch ein Leerzeichen getrennt. Alles nach dem Parameter-Feld ist ein Kommentar. Oft verwendete Operand-Statements sind zB.:

- `JOB`: markiert den Anfang eines Jobs
- `EXEC`: bezeichnet Prozedur, die ausgeführt werden soll
- `PROC`: Adresse der Prozedur
- `DD`: bezeichnet die zu benutzenden Dateien/Datasets (Data Definition)

Darauf folgt eine Parameterliste, wobei die Parameter lediglich durch ein Komma getrennt sind (ohne Leerzeichen). Die Spalten 72 - 80 werden von JCL nicht berücksichtigt. Wenn die Parameterliste über Spalte 71 hinausgeht, muss diese auf 2 Zeilen aufgeteilt werden. Dazu ist eine neue Zeile (begonnen mit „//“) zu erstellen, die die Parameterliste zwischen Spalte 4 und 16 fortführt.

Für jedes Dataset, das von einem Programm verwendet oder erstellt wird, muss ein `DD`-Statement angegeben werden. Zum Bsp.:

```
//PAY␣DD␣DSN=HLQ.PAYDS,DISP=NEW␣VENDOR␣PAYROLL
```

Unnützes Wissen: Der Unix-Befehl `dd` ist durch den JCL-DD-Operator inspiriert.

Das Namensfeld enthält den 1-8 stelligen Namen `PAY`, der als `ddname` bezeichnet wird. Über diesen `ddname`n können andere Programme, Skripte oder das Betriebssystem darauf verweisen. Das Parameterfeld enthält 2 Keywordparameter:

- `DSN`: physischer Name des Datasets
- `DISP`: Status (Disposition) des Datasets vor/nach dem Job (`NEW`: wird neu angelegt)

[5] **J**ob **E**ntry **S**ubsystem

`VENDOR PAYROLL` wird als Kommentar gewertet. Für den DD-Operator gibt es noch andere Keywordparameter, die das Dataset umfassend beschreiben. Zu den JCL-Operatoren und dem allgemeinen Aufbau bietet der „JCL Quick Guide“ von `tutorialspoint`[6] eine gute Grundlage.

Um nun die zwei Schritte Compile sowie Link Edit durchzuführen dient folgendes JCL:

```
//PRAKXXXC␣JOB␣(),CLASS=A,MSGCLASS=M,MSGLEVEL=(1,1),NOTIFY=&SYSUID,
//␣␣␣␣␣␣␣␣␣␣REGION=4M
//STEP1␣EXEC␣IGYWCL
//COBOL.SYSIN␣DD␣DSN=&SYSUID..TEST.COB(COB01),DISP=SHR
//LKED.SYSLMOD␣DD␣DSN=&SYSUID..TEST.LOAD,DISP=SHR
//LKED.SYSIN␣DD␣*
␣NAME␣COB01(R)
/*
```

`&SYSUID` wird durch den JCL-Interpreter durch die eigene Nutzerkennung ersetzt

Das erste Statement in einem JCL-Script ist immer ein JOB-Statement. Es ist üblich die TSO-Benutzer-ID plus einen angehängten Buchstaben als Label für das Job-Statement zu verwenden. Das hierfür vorgesehene Feld hat eine Länge von maximal 8 Zeichen. Aus diesem Grund haben TSO-Benutzer-ID's eine maximale Länge von 7 Zeichen.

Das EXEC-Statement enthält die Anweisung, die Prozedur `IGYWCL` abzuarbeiten. `IGYWCL` ist ein von z/OS zur Verfügung gestelltes Script (Catalogued Procedure), dass

`IGYWCLG` führt ein Compile, Link *und* GO durch.

- den COBOL-Compiler aufruft
- anschließend den Linkage-Editor aufruft
- den zu übersetzenden Quelltext von `COBOL.SYSIN` erwartet
- das erstellte Load Module unter `LKED.SYSLMOD` abspeichert
- das unter `LKED.SYSIN` angegebene Modul `COB01` zu einem ausführbaren Programm zusammensetzt

1.4 Ausführen der Skripte und des Programms

JES dient dazu, Stapelverarbeitungsaufträge (Jobs) auf die einzelnen CPUs zu verteilen und der Reihe nach abzuarbeiten. Jobs werden dem Subsystem in der Form von JCL-Scripten zugeführt, wobei deren erstes JCL-Statement ein JOB-Statement sein muss.

Das Kommando `SUB` für `submit` im ISPF-Editor bewirkt, dass der in JCL beschriebene Job des geöffneten Dokuments in die Warteschlange der von JES abzuarbeitenden Aufträge eingereiht wird.

z/OS gestattet es grundsätzlich, Programme entweder interaktiv im Vordergrund – unter TSO – oder als Stapelverarbeitungsprozesse – durch JES – im Hintergrund abzuarbeiten. Ersteres garantiert bessere Antwortzeiten, letzteres führt zu einem besseren Durchsatz.

[6] `http://www.tutorialspoint.com/jcl/jcl_quick_guide.htm`

Der JCL-Interpreter überprüft die Syntax des Scriptes und falls er keinen Fehler findet, übergibt er den Job zur Abarbeitung an das JES-Subsystem. Dies wird durch

```
IKJ56250I JOB PRAKXXXC(JOBXXXXX) SUBMITTED
***
```

bestätigt. In diesem Infotext wird dem Benutzer unter anderem die Jobnummer mitgeteilt, die für eine spätere Fehleranalyse benutzt werden kann. Wenn man nun die Eingabetaste drückt und der Job bereits durchgelaufen ist erscheint die Bestätigung:

Wenn die Bestätigung nicht sofort erscheint, kurz warten und nochmals Eingabe drücken.

```
15.10.01 JOBXXXXX $HASP165 PRAKXXXC ENDED AT N1  MAXCC=0 CN(INTERNAL)
***
```

Es wird ein `MAXCC`[7] von 0 angezeigt. Das bedeutet, dass während der Bearbeitung des Jobs keine Fehler oder Warnungen aufgetreten sind. Bei einem `MAXCC=4` sind Warnungen entstanden, bei 8 Fehler und bei 16 schwerwiegende Fehler.

Sollten nach der Eingabe ein Prompt erscheinen, der wiederholt nach einem **Data Set Name** fragt, kann die Eingabe mittels **PF1**-Taste abgebrochen werden.

Um das soeben compilierte Programm aufzurufen gibt man unter `Command` oder `Option` „`tso call 'prakXXX.test.load(cob01)'`“ ein und oberhalb der Kommandozeile erscheint:

```
Hallo Welt, unser erstes TSO-PROGRAMM in COBOL
***
```

2 Fehlersuche – SDSF

Sollte es bei einem Job zu Fehlern kommen, bietet die SDSF[8] eine Sammlung von Werkzeugen, die bei der Fehlersuche behilflich sein können. Über das `ISPF Primary Option Menu` unter `m` gelangt man zum `IBM Products Panel`. Dort lässt sich SDSF über 5 öffnen.

Durch den Command `tso sdsf` lässt sich SDSF auch direkt aufrufen.

```
  Display  Filter  View  Print  Options  Help
------------------------------------------------------------------------------
HQX7730 -----------------  SDSF PRIMARY OPTION MENU  -------------------------

DA     Active users
I      Input queue
O      Output queue
H      Held output queue
ST     Status of jobs

SE     Scheduling environments

END    Exit SDSF

Licensed Materials - Property of IBM
```

[7] **Max**imal **C**ondition **C**ode
[8] System Display and Search Facility

```
5694-A01 (C) Copyright IBM Corp. 1981, 2006. All rights reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

COMMAND INPUT ===> _                                          SCROLL ===> PAGE
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP      F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

JES speichert eine Reihe von Zwischenergebnissen in temporären Dateien ab. Ursprünglich wurden diese Datasets auf Magnetband geschrieben (Spool), um dann bei Bedarf offline ausgedruckt zu werden. Deshalb werden diese Dateien als Spool Files bezeichnet. Gibt man nun `st` unter `Command Input` ein, gelangt man zu den Statusinformation des Jobs mit der beim Submit des JCLs zurückgelieferten ID:

```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF STATUS DISPLAY ALL CLASSES                        LINE 1-3 (3)
NP   JOBNAME  JobID    Owner    Prty Queue      C  Pos  SAff  ASys Status
     PRAKXXX  TSUXXXXX PRAKXXX    15 EXECUTION          SYS1  SYS1
     PRAKXXXC JOBXXXXX PRAKXXX     1 PRINT      A   175

COMMAND INPUT ===> _                                          SCROLL ===> PAGE
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP      F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Die ersten 4 Spalten werden als NP (Non Protected) bezeichnet und dienen der Eingabe von Befehlen. Mit `s` lassen sich alle Dateien nacheinander anzeigen. Da dies allerdings sehr unübersichtlich sein kann, empfiehlt es sich statt dessen ein `?` einzugeben, was zu der Übersicht der einzelnen Spool Files führt:

```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB PRAKXXXC (JOBXXXXX)    LINE 1-5 (5)
NP   DDNAME   StepName ProcStep DSID Owner    C Dest          Rec-Cnt Page
     JESMSGLG JES2                2 PRAKXXX  M LOCAL              14
     JESJCL   JES2                3 PRAKXXX  M LOCAL              71
     JESYSMSG JES2                4 PRAKXXX  M LOCAL              61
     SYSPRINT STEP1    COBOL    102 PRAKXXX  M LOCAL             116
     SYSPRINT STEP1    LKED     103 PRAKXXX  M LOCAL             128

COMMAND INPUT ===> _                                          SCROLL ===> PAGE
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP      F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Auf diesem Screen sieht man die 3 Datasets:

- `JESMSGLG`: JES2 Message Log File
- `JESJCL`: JES2 JCL File (nach Auflösung der Systemverweise)

- `JESYSMSG`: JES2 System Messages File

Und je ein Dataset zu den ausgeführten Unterprogrammaufrufen Link und Compile.

Wenn man nun auf den vorherigen Screen zurückgeht und alle Datasets aneinandergereiht betrachtet, finden sich in den letzten 3 Screens (scrollen) meistens die Fehlercodes und Zusammenfassungen.

Das JESYSMSG Listing enthält **Memory Allocation und Cleanup** Messages. ALLOC besagt, welche Devices und wie viel Hauptspeicherplatz für den Job Step belegt wurde. Es wird ebenfalls die von dem Step verbrauchte CPU Zeit angezeigt.

```
                    ***** E N D  O F  R E P O R T *****



z/OS V1 R8 BINDER     15:10:00 MONDAY NOVEMBER 17, 2014
BATCH EMULATOR  JOB(PRAKXXXC) STEP(STEP1   ) PGM= HEWL      PROCEDURE(LKED    )
IEW2008I 0F03 PROCESSING COMPLETED.  RETURN CODE =  0.

 ----------------------
 MESSAGE SUMMARY REPORT
 ----------------------
  TERMINAL MESSAGES      (SEVERITY = 16)
  NONE

  SEVERE MESSAGES        (SEVERITY = 12)
  NONE

  ERROR MESSAGES         (SEVERITY = 08)
  NONE

  WARNING MESSAGES       (SEVERITY = 04)
  NONE

  INFORMATIONAL MESSAGES (SEVERITY = 00)
  2008  2322


  **** END OF MESSAGE SUMMARY REPORT ****
```

Anhand dieser Übersicht lässt sich schnell feststellen, wo ein Fehler aufgetreten ist. Daraufhin wird in dem dazugehörigen Spool File nach Fehlermeldungen gesucht.

3 Aufgaben

Ersetzen Sie `PRAKxxx` durch ihre eigene Nutzerkennung!

Aufgabe 1

Legen Sie zwei neue Datasets `PRAKxxx.TEST.COB` und `PRAKxxx.TEST.CNTL` mit den aus der letzten Aufgabenserie bekannten Parametern an.

Legen Sie nun ein neues Dataset `PRAKxxx.TEST.LOAD` an, dass unter `Record format` „`U`“ für Undefined statt „`Fixed Block`“ aufweist(, da es die Maschninenbefehle aufnehmen soll).

Erstellen Sie einen Screenshot von `DSLIST`, der die soeben erstellten Datasets zeigt.

Ergebnis: Screenshot `1_DSLIST.jpg` (oder png) (maximal 2 Punkte)

Aufgabe 2

Verfassen Sie ein eigenes funktionsfähiges COBOL-Programm (keine Modifikation des vorgegebenen Hallo-Welt-Programms) und legen Sie den Sourcecode in `PRAKxxx.TEST.COB(COBn)` ab (`n` sei eine Ziffer Ihrer Wahl). Das angepasste JCL-Script legen Sie bitte in `PRAKxxx.TEST.CNTL(COBSTAn)` ab.

Erstellen Sie je einen Screenshot von Ihrem ISPF-Fenster mit dem Sourcecode Ihres Programms, sowie mit der Ausgabe Ihres COBOL-Programms. Erzeugen Sie ebenfalls einen Screenshot, der das von Ihnen modifizierte JCL-Script zeigt.

Beschreiben Sie in Ihrer Abgabe kurz, was Ihr Programm macht, falls Fehler auftreten dokumentieren Sie diese (vgl. Aufgabe 3) und korrigieren Sie sie.

Ergebnisse:

- Beschreibung des Programms
- Screenshot `2_COBRUN.jpg` (oder png)
- Screenshot `2_JCL.jpg` (oder png)

(maximal 7 Punkte)

Aufgabe 3

Testen Sie, wie sich Fehler in den Spool Files bemerkbar machen. (Falls Sie keinerlei Fehler hatten, produzieren Sie welche ;))

Erstellen Sie einen Screenshot in SDFS, bei dem mindestens 1 Fehler angezeigt wird.

Was *w*ürde passieren, wenn sie Ihren laufenden `TSUXXXXX`-Job purgen *w*ürden?

Ergebnisse:

- Screenshot `3_Fehler.jpg` (oder png)
- Beantwortung der Frage (mit Begründung)

(maximal 2 Punkte)

Purgen Sie ihre abgesendeten Jobs (nur `JOBXXXXX`) aus den Spool Files, wenn Sie die Aufgaben erledigt haben. (Durch `p` vor dem Job oder „von `//p` bis `//`“, wenn mehrere Jobs gepurged werden sollen.)

Abgabe:
Erstellen Sie eine `pdf`-Datei in das Sie **alle** Ergebnisse zu den Aufgaben einfügen und laden Sie diese bei Moodle in den Abgabeordner hoch.

Dateiname: `ECG2_<Namen beider Teammitglieder>`

Für das Erreichen der maximalen Punktzahl müssen bei der Praktikumsabgabe neben der vollständigen Bearbeitung der Aufgaben, auch Kenntnisse und Verständnis der zugrunde liegenden Themen dieses Arbeitsblattes bestätigt werden.

4 Quellen

Bei diesem Dokument handelt es sich um eine grundlegende Überarbeitung des alten Tutorials (Tut02) zu der Vorlesung „Einführung in z/OS“ der Universität Leipzig und Tübingen.

Die Aufgaben-Verschönerung wurde von `http://www.texample.net/tikz/examples/framed-tikz/` übernommen.

Die Icons aus Abbildung 1 entstammen dem „Humanity“-Theme von Gnome.