ANDREAS PROUZA

Leonard-Bernstein-Straße 8/2/17.07
1220 Wien / Österreich

Mail: andreas@prouza.at
Web: https://www.prouza.at

SNDMAIL

on IBM i

1 Überblick

Der Mail Client besteht aus 2 Teilen:

- RPG-Welt
  Das Ganze „rundherum“
  - Command für den Aufruf
  - Programme um die Daten in die Tabellen einzutragen
  - Mail Service welches die Liste der Mails abarbeitet und an das Java übergibt
  - Data queue Konzept zur Kommunikation mit dem Mail Service (Sync/Async)
- Java
  Der eigentliche Mailversand

2 Mindestanforderungen

Java 1.8

Der Java Client ist mit der Version 1.8 erstellt worden.

Sollte es nötig sein eine ältere Version zu unterstützen muss nur geprüft werden ob die Abhängigkeiten ebenfalls mit der älteren Version kompatible sind.

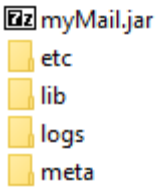
IBM i 7.2

3 Java

Im IFS wird ein Verzeichnis hinterlegt, mit allen nötigen Ordnern und Files.
Im Setup ist wird folgendes Verzeichnis angelegt:

/usr/prouza/mail4i

Dieses kann geändert werden. Dafür muss man lediglich auch den Property Eintrag in der Tabelle PROUZALIB/PROPERTIES anpassen (siehe 4.1.3 PROPERTIES)



- myMail.jar
  Das Java File welches die Klassen und Funktionen beinhaltet, die vom RPG-Part aufgerufen werden
- etc/
  Konfigurations Verzeichnis
- log.properties
  Einstellungen für den Logger wo welche Logs geschrieben werden
- simplejavamail.properties
  Mail Server Einstellungen
- lib/
  Hier befinden sich alle Java Files die für diese Applikation benötigt werden.
- logs/
  Das Log-Verzeichnis in dem die Log Files geschrieben werden.
  Dies wird im etc/log.properties definiert
- meta/
  Hier werden der HTML Body und Plain Text Body ins IFS geschrieben.
  Ebenfalls gibt es ein JSON File welches alle Informationen zur Mail beinhaltet (Empfänger, Betreff, wo sich die Attachments befinden usw.)
  Dieses Meta File wird vom Java Programm geladen und ist die Basis für den Mailversand.

3.1 Meta File

Das JSON basierte Meta File beinhaltet alle Daten aus den Tabellen.
Dadurch ist es für den Java Client nicht nötig auf die Datenbank zugreifen zu müssen.

Das Meta File setzt sich aus der Mail-Id und der Endung ".json" zusammen. (Z.B.: 29.json)

```
{
  "id": 29,
  "to": "andreas@prouza.at",
  "cc": "office@customer.com",
  "subject": "Test Mail",
  "priority": "*NORMAL",
  "html": "/usr/prouza/mail4i/meta/29_body.html",
  "text": "/usr/prouza/mail4i/meta/29_body.txt",
  "attachment": [
    {
      "ifs": "/home/prouza/test1.pdf"
    },
    {
      "ifs": "/home/prouza/test2.gif"
    }
  ]
}
```


Wird im Command der Text für HTML und/oder Plain Text als String mitgegeben, so wird der Inhalt als File ebenfalls ins IFS in das meta/ Verzeichnis geschrieben.

4 RPG-Welt

4.1 DB2

3 Tabellen werden verwendet:

4.1.1 MAILCLIENT

Hier werden alle Mails für die weitere Verarbeitung eingetragen

4.1.2 MAILATTACH

Alle Attachments einer Mail. Von den Attachment-Files selbst wird lediglich der IFS Pfad hier eingetragen.

4.1.3 PROPERTIES

Eine allgemeine Tabelle für alle Applikationen in der PROUZALIB.
Hier werden diverse Settings für die entsprechende Applikation hinterlegt.

MAIL_MAX_TRIES
Wie oft maximal eine Mail versucht wird zu versenden.

MAIL_IFS_ROOT
Das Ausgangsverzeichnis der Java Applikation im IFS.
Dies kann beliebig geändert werden.

4.2 RPG

4.2.1 Direktaufruf der Prozedur

Hierfür sollte das Bindeverzeichnis PROUZALIB/PROUZADIR ebenfalls eingebunden werden.

Neben dem Command können Mails auch direkt über die RPG Prozedur versendet werden.

In der Copy-Strecke PROUZALIB/QMAILSRC(SNDMAILC) sind die nötigen Deklarationen:

```
dcl-pr sendMail int(20);
  P_TO       varchar(3302) const options(*varsize : *trim);
  P_CC       varchar(1586) const options(*varsize : *omit : *trim);
  P_BCC      varchar(1586) const options(*varsize : *omit : *trim);
  P_SUBJECT  varchar(512)  const options(*varsize : *trim);
  P_TEXT     varchar(3000) const options(*nopass : *omit : *varsize : *trim);
  P_HTML     varchar(3000) const options(*nopass : *omit : *varsize : *trim);
  P_FROMIFS_TEXT  varchar(256)  const options(*nopass : *omit : *varsize : *trim);
  P_FROMIFS_HTML  varchar(256)  const options(*nopass : *omit : *varsize : *trim);
```

```
  P_ATTIFS    likeds(attach_ds_tmp) dim(MAX_ATTACHMENTS)
                            const options(*nopass : *omit : *varsize : *trim);
  P_SENDER    varchar(128)  const options(*nopass : *omit : *varsize : *trim);
  P_REPLYTO   varchar(128)  const options(*nopass : *omit : *varsize : *trim);
  P_PRIORITY  varchar(7)    const options(*nopass : *omit : *varsize : *trim);
  P_ASYNC     ind           const options(*nopass);
END-Pr;
```

4.2.2 Mail Service

Das Mail Service läuft im Batch als eigener Job (Endlosschleife).

Der Job kann aber mit ENDJOB und *CNTRLD einfach beendet werden.
Da der Job in periodischen Abständen prüft ob ein Ende-Signal übermittelt wurde (z.B. ENDJOB oder PWRDWNSYS).
Dadurch ist ein kontrolliertes Beenden des Jobs jederzeit möglich.

Meine Empfehlung für SBMJOB:

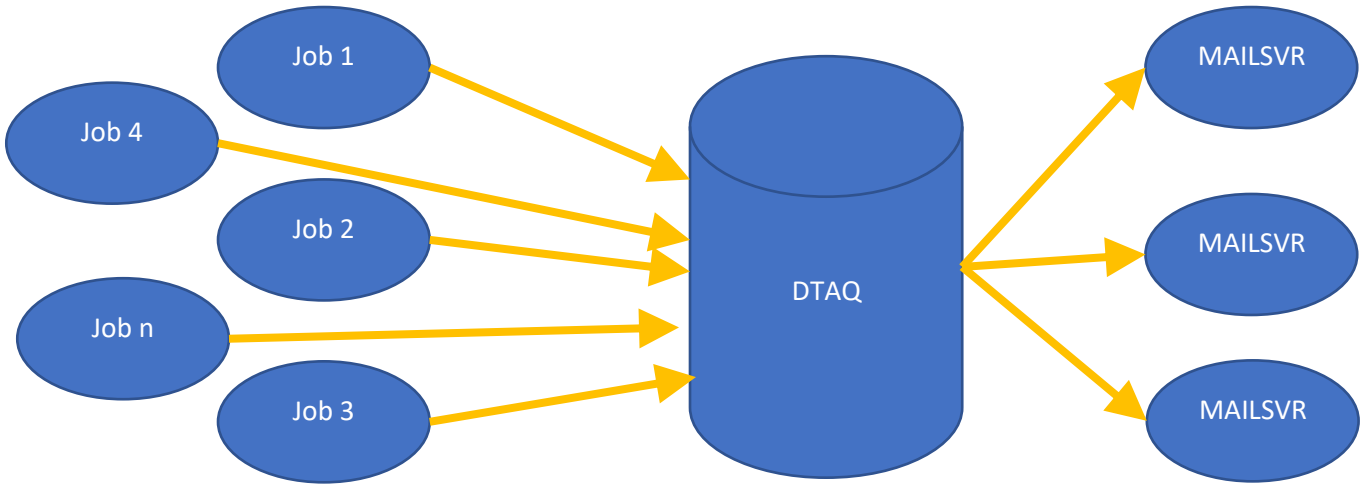
SBMJOB CMD(CALL PGM(PROUZALIB/MAILSVR)) JOB(MAILSVR) JOBQ(QS36EVOKE) CURLIB(PROUZALIB) JOBMSGQFL(*WRAP)

JOBQ
Hier verwende ich meistens die QS36EVOKE, da diese auf allen Maschinen immer aktiv ist und die JobQ mit maximaler Anzahl paralleler Jobs mit *NOMAX gesetzt ist.
Die QBATCH erlaubt per Default nur 1 Job aktiv und der ist oft schon besetzt.

JOBMSGQFL
Es werden periodisch JOBLOG Ausgaben getätigt. Dadurch sieht man was verarbeitet wurde und ob der Job noch arbeitet.
Da dieser Job permanent läuft kann es dadurch zu einem Überlauf kommen.
Mit *WRAP stelle ich sicher, dass bei einem Überlauf die älteren Einträge einfach gelöscht werden.

Parallelisierung des Mail-Services

Es können zur Laufzeit auch mehrere Jobs parallel gestartet werden. Da es Data Queue basiert ist, kann es beliebig jederzeit nach oben oder unten skaliert werden.



Soll heißen: Das MAILSVR Programm, ist so aufgebaut, dass es immer dieselbe Data Queue abarbeitet.

Werden mehrere Jobs via SBMJOB gestartet, greifen alle Jobs auf dieselbe Data Queue zu.

Dadurch kann die Verarbeitung der Data Queue (welche die zu sendenden Mail-Id's beinhaltet) parallel abgearbeitet werden.

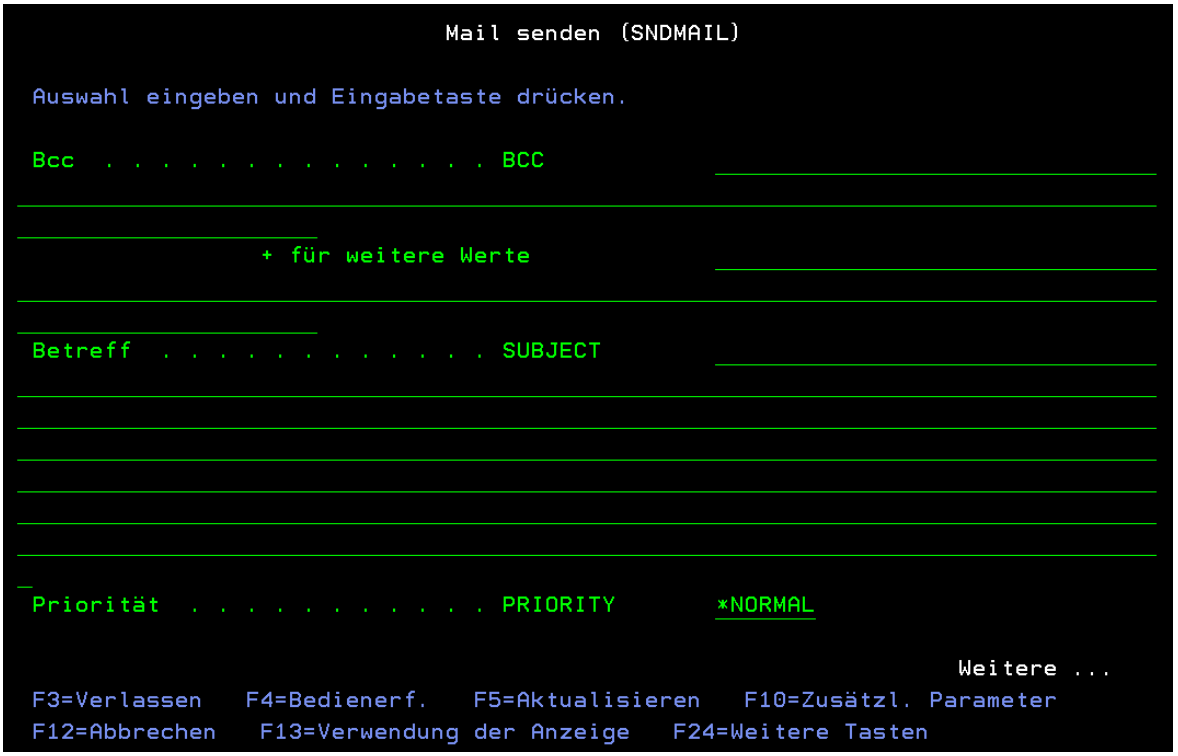
Dies kann gerade dann sinnvoll sein, wenn die Kommunikation mit dem Mailserver grundsätzlich etwas "länger" dauert, es aber zu Spitzenzeiten eine Vielzahl an Mails in der Minute verschickt werden soll.

Möchte man die Zahl der Jobs wieder dezimieren, so ist dies jederzeit einfach mit einem ENDJOB möglich.
Der Job beendet sich von selbst automatisch zum nächst möglichem Zeitpunkt. Z.B. sobald die aktuelle Verarbeitung/Versand einer Mail beendet wurde.

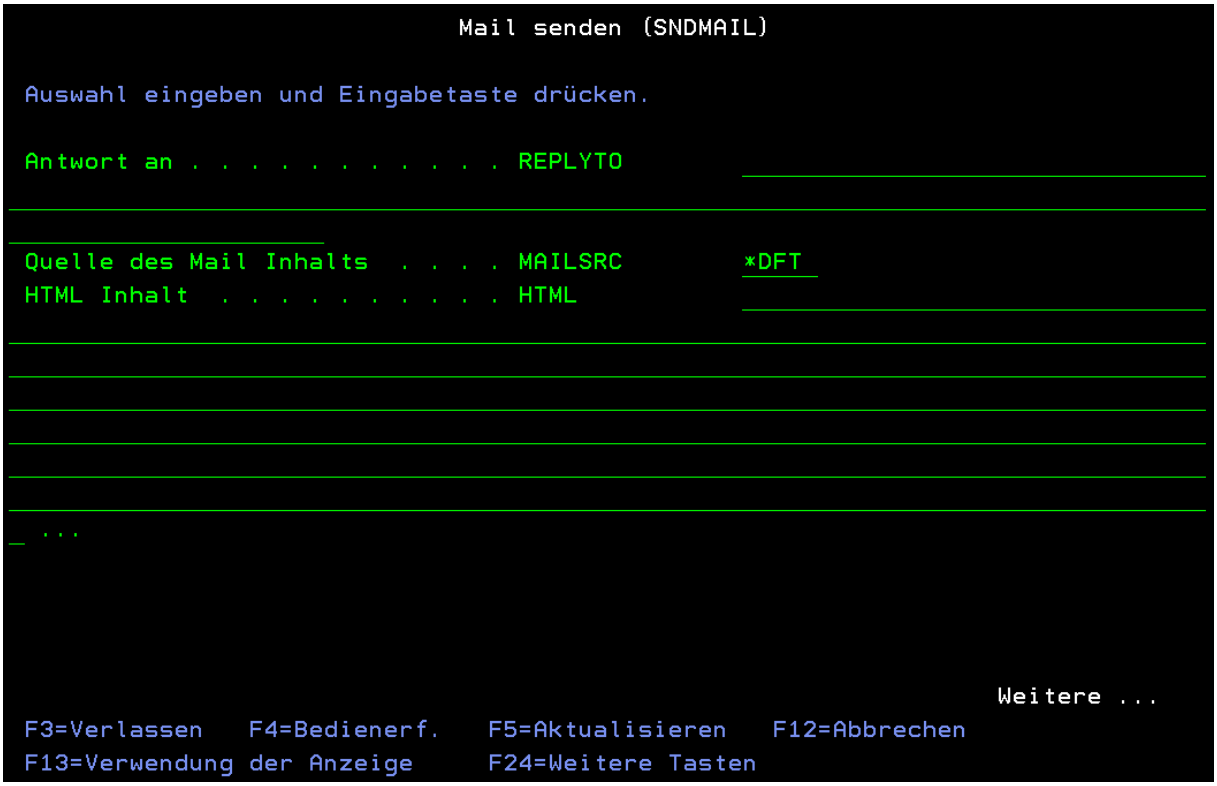
4.2.3 SNDMAIL Command

Sender:

Absender-Mail Adresse. Wenn dieser Parameter nicht gesetzt wird, wird der Absender aus dem Konfigurationsfile genommen.



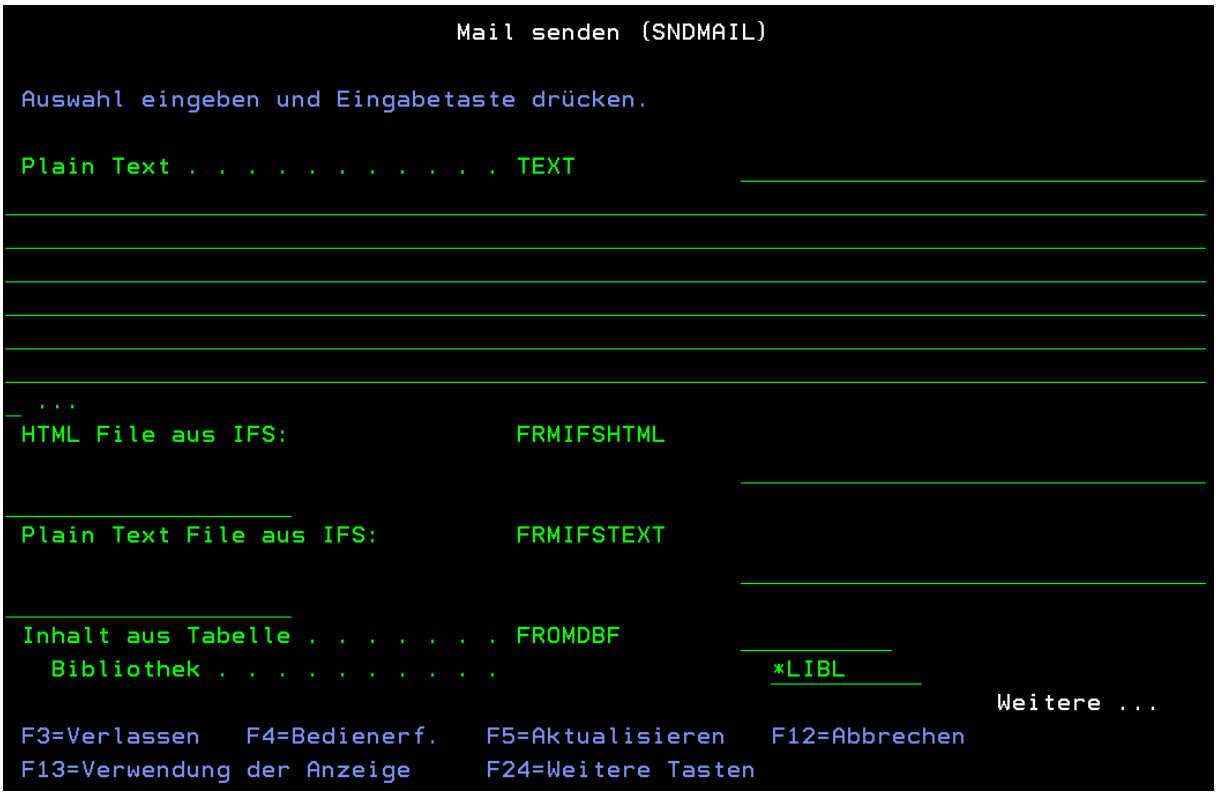
Prioritäten: *NORMAL, *HIGH, *LOW



REPLYTO: An welche Mail-Adresse eine Antwort geschickt werden soll.

MAILSRC: *DFT, *SPL, *IFS, *DBF

- *DFT
  HTML Body und Plain Text Body werden vom Parameter HTML und TEXT genommen.
- *SPL
  Plain Text Body wird vom Parameter SPLFILE, SPLJOB & SPLSNBR genommen.
- *IFS
  HTML Body wird vom Parameter FRMIFSHTML und der Plain Text Body vom Parameter FRMIFSTEXT genommen.
- *DBF
  Der Plain Text Body wird vom Parameter FROMDBF genommen.



Hier siehe Parameter MAILSRC

ATTIFS

Aktuell können über den Command maximal 50 Attachments mitgegeben werden.

Mit der Prozedur addMailAttachment(mail-Id : attachment-DS) können beliebig viele Attachments einer Mail zugeordnet werden.

Bei jedem Attachment kann, neben dem IFS Pfad, optional auch folgende Angaben mitgegeben werden:

- Content-Id
  wenn z.B. ein Bild in den HTML Body eingebunden werden soll
- Content Type
  im MIME Type Format (image/gif, application/pdf, application/msword, application/zip, …)
- Komprimierung

Wird Content Type nicht mitgegeben, so wird dieser automatisch aus dem File selbst ermittelt.

ASYNC

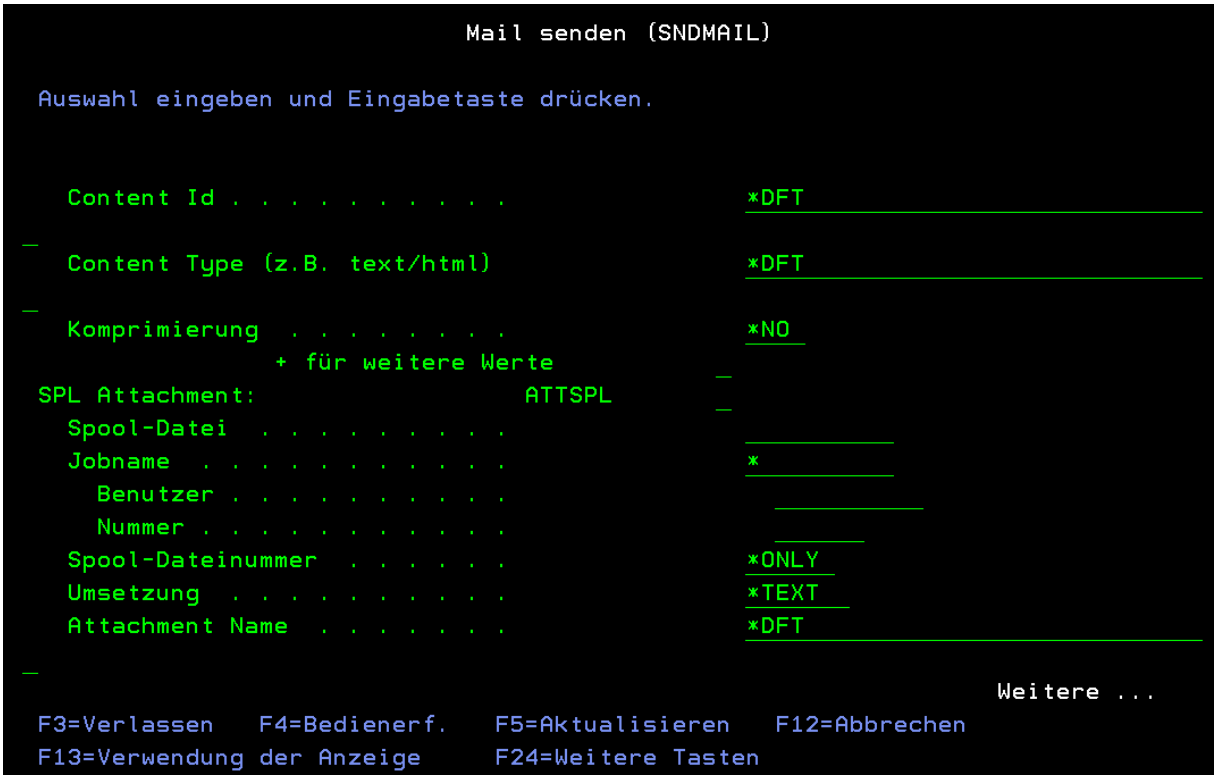
- *YES
  Es wird nicht auf Antwort vom Mail Service gewartet. Die Verarbeitung geschieht asynchron (parallel).
- *NO
  Der Command wird erst nach Erhalt einer Rückantwort vom Mail Service beendet.

Per Default ist *YES hinterlegt, da es beim Versenden von Mails wenig Sinn macht, dass die Applikation wartet bis die Mail vom Mailserver entgegengenommen hat.

ATTDBF

Aktuell wird hier die Tabelle lediglich als Text File im Attachment angehängt.

Da es ebenfalls ein Attachment ist, gibt es auch hier wieder die Möglichkeit zusätzlich Content-Id und Content-Type mitzugeben.



Spool Files können ebenso wie Tabellen als Attachment mitgegeben werden.

4.2.4 Beispiele

Mail Body wird direkt im Command übergeben

```
SNDMAIL TO('andreas@prouza.at') SUBJECT(test) HTML('<h1>Das
ist der HTML Body</h1>') TEXT('Das ist der normale Plain
Text')
```

Mail Body ist in einem IFS File hinterlegt:

```
SNDMAIL TO('andreas@prouza.at') SUBJECT(test) MAILSRC(*IFS)
FRMIFSHTML('/home/prouza/mailbody.html')
FRMIFSTEXT('/home/prouza/plainbody.txt')
```

Mail Body aus einer Tabelle/View

Der Inhalt der View oder Tabelle wird als Plain Text gesendet.

```
SNDMAIL TO('andreas@prouza.at') SUBJECT(test) MAILSRC(*DBF)
FROMDBF(PROUZALIB/QMAILSRC)
```

Mail Body aus einem Spool File

Der Inhalt des Spool File wird als Plain Text gesendet.

```
SNDMAIL TO('andreas@prouza.at') SUBJECT(test) MAILSRC(*SPL)
SPLFILE(QPRINT) SPLJOB(011832/PROUZA/QPADEV0001) SPLSNBR(2)
```

Kombination aus Tabelle/View & Attachments

Der Mail Body wird von einer Tabelle/View bezogen und zusätzlich noch weitere IFS Files als Attachment angehängt.

```
SNDMAIL TO('andreas@prouza.at') SUBJECT(test) MAILSRC(*DBF)
FROMDBF(PROUZALIB/QMAILSRC)
ATTIFS(('/usr/prouza/mail4i/logs/log_MailProcess.log')
('/home/prouza/out10.pdf'))
```