

—BONUSKAPITEL—

Daten in
Dateien,
Datenablage
ohne DB

Mit Schirm, Charme – und vielleicht noch einem Melonensorbet

»Warst du schon einmal bei einem richtig guten Italiener?« Schrödinger ahnt nichts Gutes.

»Also so einem, wo du bereits am Eingang deine Garderobe abgeben kannst.« – »Ja, das Prinzip der Garderobe ist mir bekannt, wieso?« – »Naja, wäre es nicht klasse, wenn die User deiner Anwendungen in ABAP auch Dateien einlesen oder herunterladen – also an der Garderobe abgeben könnten?« Au Backe, das ist ja mal eine miese Metapher. Aber es ist vielleicht trotzdem wichtig zu wissen, wie man mit Daten umgeht, wenn man kein Datenbanksystem hat.

Daten ohne Datenbank

Roland, Roland, ich war gerade in der Kaffecke und hab schon richtig mit meinem ABAP-Wissen punkten können. Da hab ich einen Wunsch aufgeschnappt, dass Daten auch in Dateien abgespeichert werden sollen. Wie geht denn das?

Gratuliere, wenn das keine Motivation für mehr ist! Natürlich kannst du **Daten** auch in Dateien ablegen. Da gibt es einige Möglichkeiten, zum Beispiel auf den **Applikationsservern** oder **Clients** der User.

Präsentationsschicht

Hier liegen Daten in Files,
am Client.

Applikationsschicht

Hier liegen Daten auch in Files,
aber am Applikationsserver.

Datenbankschicht

Na und hier liegen die Daten in Tabellen,
am Datenbankserver.

Die Daten liegen an verschiedenen Orten. Um es dir zu zeigen, hab ich das olle Drei-Schichten-Modell herausgekratzt und abgestaubt. Manche User hätten am liebsten alles auf »ihrem« Rechner. Darum bieten wir ihnen auch Up- und Download-Funktionen an. Damit die Daten auch zentral im Dateisystem abgelegt werden können, werden uns Funktionen für Dateioperationen auf dem Applikationsserver angeboten. Daten in der Datenbank, das ist eigentlich das Natürlichste auf der Welt, oder?

Du kannst die Daten auch in bestimmte **Formate** transformieren und Client-Programme aufrufen, die die Daten visualisieren. Ich zeige dir ein paar Möglichkeiten.

GUI-Loads – Upload und Download

Diese User wollen die Daten, und ihnen reicht nicht die Datenbank. Naaaaaein, die Daten muss man angreifen können. Am besten **downloaden** in ein Format, das die üblichen (Programm-)Verdächtigen darstellen können, und dann ausdrucken. Tausende Seiten, fein säuberlich am Drucker gestapelt und dann niemals abgeholt.

Datenfriedhof Ausdruck.



**SOOOOOOOO
enden Datent!**

Das Bedürfnis können wir gerne befriedigen. Da hilft uns die SAP natürlich und stellt uns die Klasse **cl_gui_frontend_services** zur Verfügung, mit der wir sehr einfach aus unserem Programm heraus die Daten zu den Usern bringen.

*Ja, wie, aus der Datenbank raus, in "eigene Dokumente" rein?
In welcher Struktur denn? [Feld-Komma-Feld-Komma-Feld-
Neue Zeile-Feld-Komma], oder was ...?]*

Es wäre nicht die SAP, hätte sie sich nicht einiges dazu überlegt.

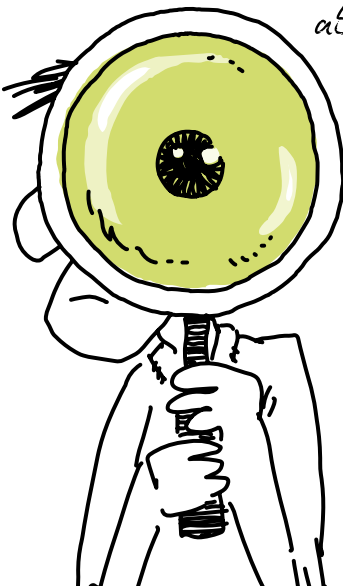
Ich werde dir Beispiele zu folgenden Themenbereichen zeigen:

- ☛ **Verzeichnisfunktionen**, zum Beispiel die Methode **get_temp_directory**. Damit stehen dir Operationen für Verzeichnisse zur Verfügung.
- ☛ **Clipboard schreiben/lesen**, zum Beispiel die Methode **clipboard_export**. Damit kannst du Daten ins Clipboard kopieren und in anderen Anwendungen wieder einfügen, wie zum Beispiel dem Notepad.
- ☛ **Datei-Up-/Download**, also die Methoden **gui_download** und **gui_upload**. Damit kannst du Daten in Client-Dateien schreiben und aus diesen lesen.

Das war noch nicht alles. Die Klasse bietet auch noch Methoden zu folgenden Themen:

- ☛ Environment
- ☛ Dateifunktionen
- ☛ Abfragefunktionen, zum Beispiel Windows-Directory, Windows-Version etc.
- ☛ Standarddialoge (Öffnen, Speichern, Verzeichnisauswahl)

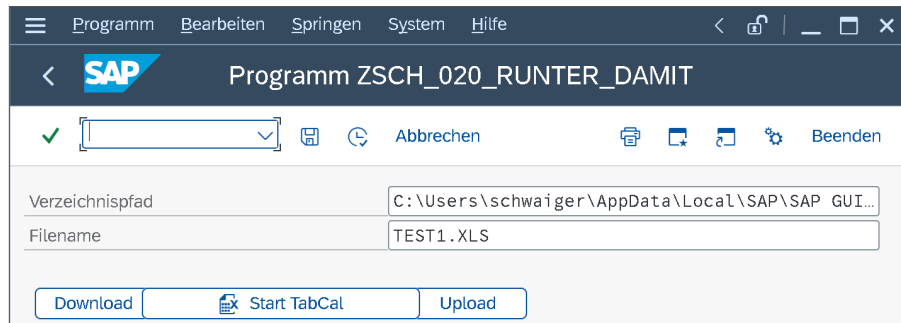
Fokus, Fokus. Wir wollen uns auf Daten konzentrieren, die auf dem Client abgelegt sind.



Ich bin ja schon fokussiert wie ein Laserstrahl. Vorab könntest du dich schon mal mit der Klasse **cl_gui_frontend_services** vertraut machen, aber natürlich nur, wenn du gerade vor einem SAP-System sitzt, so wie ich jetzt gerade. In der Tat habe ich gerade die **SE24** aufgerufen und schmökere durch die Methoden der **cl_gui_frontend_services**.

Frontend Services

Ich fange für mich mal ganz einfach an, um noch eine kleine Schaffenspause einlegen zu können, und verpasse dir, 1-2-3, eine Aufgabe, dideldei. Wir werden uns mit dem **Download** und dem **Upload** von **Daten auf dem Client** beschäftigen.



So sollte das Selektionsbild aussehen.

Motivation = ausreichend.

Das ist ja kein Ponyhof hier.



[Schwierige Aufgabe]

Du sollst in der folgenden Aufgabe den Programmrahmen für die Implementierung der Up- und Download-Funktionalität implementieren. Lege das Programm **zsch_020_runter_damit** an, das vom Aufbau her dem Programm **zsch_019_datenmanipulator** entspricht. Im Selektionsbild gibt es zwei Parameter: **pa_dir** und **pa_file**, beide vom Typ **string**. Des Weiteren benötigen wir drei **Buttons**: einen Button für den Download (**down**), einen Button für den Aufruf einer Tabellenkalkulation (**spdsh**) und einen Button für den Upload (**up**). Orientiere dich bei der Implementierung nahezu 100 % am Manipulator.

Gleich vorweg: Ich werde dich für den Anfang mal nur geringfügig mit Informationen anfüttern.

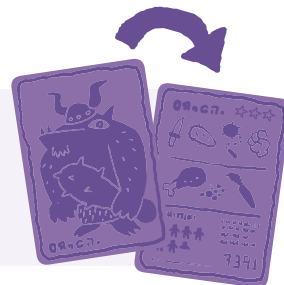
[Hintergrundinfo]

Korruptionsstrafrechtsänderungsgesetz 2012 Österreich.

Unter »Anfüttern neu« steht: »[...] Wer lediglich einen geringfügigen Vorteil annimmt oder sich versprechen lässt, ist nicht zu bestrafen, außer die Tat wird gewerbsmäßig begangen. [...]«

Eine schöne österreichische Lösung: Ein bisserl was geht immer.

Also darfst du dich nicht zu viel von mir mit Informationen anfüttern lassen.



Hast du schon den **Programmrahmen**? Hier ist meiner, und ich werde dir nur die Teile erklären, die nicht im Manipulator bereits erklärt wurden:

```
REPORT zsch_020_runter_damit.
TYPE-POOLS: abap, icon.*1
* OK-Code
TABLES sscrfields.
* Die Kunden
TYPES: gtt_customers TYPE TABLE OF zsch017customer.*2
* Verzeichnis und Dateiname
PARAMETERS: pa_dir TYPE string,*3
              pa_file TYPE string.

* Die Drucktasten
SELECTION-SCREEN SKIP 2.
SELECTION-SCREEN PUSHBUTTON 2(10) down *4
                      USER-COMMAND down.
SELECTION-SCREEN PUSHBUTTON 12(25) spdsh
                      USER-COMMAND spdsh.
SELECTION-SCREEN PUSHBUTTON 37(10) up
                      USER-COMMAND up.

* Die Kundendaten
DATA: gt_customers TYPE gtt_customers.*5
* Vorschlagswert für das Verzeichnis
DATA: gd_temp_dir TYPE string.
* Kombi aus Pfad und Datei
DATA: gd_file_path TYPE string.
```

```
INITIALIZATION.
* Text für den Button
  down = 'Download'(b01).
  up = 'Upload'(b02).
* Icon auf Button ergänzen
CALL FUNCTION 'ICON_CREATE'*6
  EXPORTING
    name      = icon_xls
    text      = 'Start TabCalc'
    info      = 'Start!'
    add_stdinf = space
  IMPORTING
    RESULT    = spdsh
  EXCEPTIONS
    OTHERS    = 0.

* Directory ermitteln, in das geschrieben wird *7
* Parameter vorbelegen
* Parameterwerte vorschlagen
  pa_dir = gd_temp_dir.*8
```

*1 Die Typgruppe **icon** werde ich verwenden, um einen **Button** nicht nur mit einem Text, sondern auch mit einem **Bildchen** zu schmücken. In dieser Typgruppe sind Konstanten für **Icons** definiert.

Oh, wie schön, das habe ich schon lange gesucht.



Start TabCal

Button mit Text und Bild.

*2 Da ich in diesem Programm mehr auf **lokale Modularisierung** achten möchte, wird ein **Typ** für die Schnittstellen der Unterprogramme benötigt, um die Daten der **Kunden** zu übergeben – **gtt_customers**. Wie nennt sich der Typ noch mal schnell?

Tabellentyp, yeah.



*3 Die zwei Parameter **pa_dir** und **pa_file**, beide vom Typ **string**, übernehmen die Eingabe des Anwenders bzw. der Anwenderin zum **Dateipfad** und **Dateinamen**. Beim Starten des Programms werden wir diese auch vorbelegen.

*4 Drei Buttons im Selektionsbild: **down**, **spdsh** und **up** zum **Runterladen**, **Tabellenkalkulation starten** und zum **Hochladen**.

*5 Die Variable **gt_customers** wird die **Daten zu den Kunden** halten, **gd_temp_dir** das **Verzeichnis**, das wir vorschlagen werden, und **gd_file_path** die Verknüpfung aus **Pfad** und **Dateiname**, also den **absoluten Pfad** zu den Daten.

*6 Und das ist schon **die erste Neuerung**. Mithilfe des Funktionsbausteins **ICON_CREATE** kannst du ein **Icon mit Text** produzieren lassen, das wir für die **Drucktaste** verwenden. Zuerst suchst du dir den **Namen des Icons** (Transaktion **ICON** ist möglicherweise hilfreich), legst die Texte fest und bekommst den Wert zurückgeliefert, den du direkt dem Button **spdsh** zuweist. Fertig ist der **Button-Aufkleber**.

*7 Hier steht nur ein kurzer Satz, der dann aber zu einem Unterprogramm umgebaut werden soll. Das sind unsere (deine) **Arbeitsaufträge**, um das Programm zu vervollständigen. Ich werde mit dir natürlich Schritt für Schritt jeden Kommentar abarbeiten. Fast wie ein Buchhalter mit Schirmkappe und Ärmelschoner, den gespitzten Bleistift nicht vergessend.

*8 An dieser Stelle belege ich die Parameter mit Werten vor. Der Dank der User ist uns gewiss.

* Dateinamen vorschlagen

```
pa_file = 'TEST1.XLS'.
```

AT SELECTION-SCREEN.

* Pfad zusammenstellen

```
CONCATENATE
```

```
pa_dir
```

```
'\'
```

```
pa_file
```

```
INTO gd_file_path.*9
```

* Drucktastenbehandlung

```
CASE sscrfields.*10
```

```
  WHEN 'SPDSH'.
```

* Tabellenkalkulation aufrufen

```
  WHEN 'DOWN'.
```

* Die Kundendaten ermitteln

* Daten ins Clipboard schreiben

* Zum Testen Notepad öffnen und Tastenkombi Strg + V

* Datei schreiben - Download

```
  WHEN 'UP'.
```

* Datei holen

```
ENDCASE.
```

So, das war mal der nahezu vollständig bekannte Rahmen. Jetzt kommen die neuen Aspekte. Ich werde mich im Programmrahmen von oben nach unten bewegen. Auf alle Fälle legen wir immer ein Unterprogramm pro Anforderung an. Im Manipulator waren wir da etwas schlampig. Kapitel 7 wird benötigt.

*9 Mithilfe von **pa_dir** und **pa_file** stelle ich den **absoluten Pfad** zusammen und lege das Ganze in der Variablen **gd_file_path** ab.

*10 Last but not least die **Drucktastenbehandlung**, so wie du sie schon vom Manipulator kennst. Da sind aber einige Punkte zu erledigen, jeder Kommentar ist ein Arbeitsauftrag.

[Belohnung]



Du hast dir eine kleine Trinkpause verdient. Es ist wichtig, dass dein Hirn immer im Saft steht. Da funktioniert das Denken besser. Optimal wäre etwas mit Omega-3-Fettsäuren. Vielleicht einen Lachs- oder Leinölcocktail? Nimm dir 30 Sekunden Zeit, in Ruhe.

mmhmm. Einfach weiter im Text, bitte.



Pfad ermitteln

Bevor wir die **Daten runterladen** können, müssen wir wissen **WOHIN!**
Dazu sollten wir den **Pfad** zur Datendatei **ermitteln**. Die Klasse **cl_gui_frontend_services** hilft dir dabei tatkräftig, zum Beispiel mit der Methode **get_temp_directory**, wie du im Folgenden erkennen kannst. Diese Methode liefert dir das SAP-Temp-Verzeichnis auf deinem Client zurück, und das, denke ich, ist ein guter Vorschlag für den Ablageort der Datei.



*Wohin soll ich das Unterprogramm stellen?
Ach, jetzt sehe ich es erst. Das ist ja die Definition.
Also ans Ende des Programms!*

```
FORM get_temp_dir CHANGING cd_temp_dir TYPE string.*1
```

```
CALL METHOD l_gui_frontend_services=>get_temp_directory *2
```

```
CHANGING
```

```
temp_dir = cd_temp_dir
```

```
EXCEPTIONS
```

```
cntl_error = 1
```

```
error_no_gui = 2.
```

```
IF sy-subrc <> 0.*3
```

```
* Fehlerbehandlung
```

```
ENDIF.
```

```
* Flush der Methoden
```

```
CALL METHOD cl_gui_cfw=>flush *4
```

```
EXCEPTIONS
```

```
cntl_system_error = 1
```

```
cntl_error = 2
```

```
OTHERS = 3.
```

```
IF sy-subrc <> 0.
```

```
* Fehlerbehandlung
```

```
ENDIF.
```

```
ENDFORM.
```

*1 Wie bereits angekündigt, verpacke ich das Lesen des Pfads in ein Unterprogramm. Über den **CHANGING**-Formalparameter **cd_temp_dir** liefere ich den Pfad an die aufrufende Stelle zurück.

*2 Die Methode **get_temp_directory** retourniert den **Pfad zum Temp-Verzeichnis auf dem Client**.

Achtung, Einschaltung des Autors:
Die Anweisung hab ich mir natürlich durch die Drucktaste **Muster** erzeugen lassen.
Ende der Einschaltung.

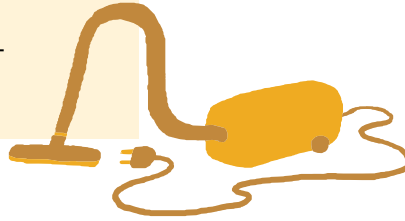
*3 Falls es zu **Ausnahmen** kommt, soll die Fehlerbehandlung noch implementiert werden. Das überlasse ich dir, zum Beispiel mit einer Fehlernachricht zu reagieren.

*4 Die Methode **cl_gui_cfw=>flush** wird benötigt, weil sie nicht in der Methode **get_temp_directory** aufgerufen wird, dies aber nötig ist, damit der Aufruf **get_temp_directory** tatsächlich **abgearbeitet** wird. Das klingt jetzt vielleicht **mystisch**, lässt sich aber sehr einfach erklären. Die Klasse **cl_gui_frontend_services** ist Teil des **SAP Control Frameworks** (Kapitel 13).

Die Ausführung von Methoden von solchen **Control-Framework-Klassen** wird immer erst am **Ende des Dynpro-PBO** angestoßen. Das wäre aber zu spät. Wir brauchen das Ergebnis genau zu diesem Zeitpunkt, und nicht erst am Ende des PBO. Daher provozieren wir den frühzeitigen Aufruf der Methode mit **cl_gui_cfw=>flush**.

[Einfache Aufgabe]

Implementiere den Aufruf des Unterprogramms zum passenden Zeitpunkt. Vorschlag: im **INITIALIZATION**-Abschnitt. Verwende die Variable **gd_temp_dir** als Aktualparameter.



Ich will den Aufruf ja gar nicht aufschreiben, den hast du sicher schon 1.000 Mal fertig, aber ich kann einfach nicht anders, ich muss schreiben.

```
* Directory ermitteln, in das geschrieben wird
PERFORM get_temp_dir
    CHANGING
    gd_temp_dir.
```

Diesen Aufruf hab ich mir zum Beispiel mit **Drag & Drop** aus der **Objektliste produzieren lassen**. Nur zur Erinnerung: **Faul sein!**



[Belohnung/Lösung]

Hast du heute schon deinen Nacken gedehnt? Noch nicht? Also entweder aufrecht sitzen oder stehen. Kopf nach vorne neigen, so weit wie es geht, und nach hinten, so weit wie es geht. Da sollte es krachen. Kopf gerade und einmal nach links schauen, so weit wie es geht, und einmal nach rechts schauen, so weit wie es geht.



Download now, zumindest vorbereitet – Download vorbereiten



[Einfache Aufgabe]

Entwickle das Unterprogramm **get_customer_data** zum Lesen der Kundendaten, und verwende den Typ **gtt_customers** zur Typisierung der Schnittstelle. Rufe das Unterprogramm an der passenden Stelle auf, und übergib den Aktualparameter **gt_customers**.

Ich bin sehr stolz auf dich. Wenn ich mir so meine Aufgabenstellungen durchlese, bin ich doch sehr beeindruckt, wie ich mit dir kommunizieren kann. Das klingt schon ziemlich **Sappisch**.

Nächster Teil: **Download**. Wenn du dir den **WHEN 'DOWN'** - Abschnitt der Drucktastenbehandlung im Programmrahmen nochmals ansiehst, dann erkennst du bereits unser Arbeitsprogramm:

- ☞ die **Kundendaten ermitteln**
- ☞ Daten ins **Clipboard schreiben** (nicht nötig für den Download, passt aber gut in das Beispiel)
- ☞ Datei schreiben – **Download**

Erstens Definition:

```
FORM get_customer_data  CHANGING
                        ct_customers TYPE gtt_customers.

  SELECT * FROM zsch017customer INTO TABLE ct_customers.
ENDFORM.
```

Zweitens AUFRUF:

```
  WHEN 'DOWN'.
* Die Kundendaten ermitteln
    PERFORM get_customer_data
          CHANGING
          gt_customers.
```

Das hast du auch so gemacht, oder? Wenn nicht, schicke bitte eine Postkarte an den Verlag mit deiner Implementierung.

Auch nett für Datenwiederverwendung – Clipboard

Es wird besser. Nun kommt der erste spannende Teil, den ich als **Extrazuckerl** für dich eingebaut habe: die Daten ins **Clipboard** schreiben.

[Zettel]

Die **Zwischenablage** (engl. Clipboard) oder Ablagemappe ist ein **Zwischenspeicherbereich**, der den einfachen Austausch von Daten (Text, Bildern und anderen Objekten) zwischen Programmen ermöglicht.

Das ist deswegen so nett, weil du mit der Tastenkombination `[Strg]+[V]` die Texte an einer beliebigen Stelle, in einem beliebigen Programm, einfügen kannst. Die Methode **cl_gui_frontend_services=>clipboard_export** erledigt das Schreiben in die Zwischenablage für uns. Sie benötigt nur die Daten. Ich bin mir ziemlich sicher, dass du das ohne meine Hilfe implementieren kannst.

[Schwierige Aufgabe]

Lege das Unterprogramm **write_to_clipboard** an, und implementiere darin den Aufruf der Methode **cl_gui_frontend_services=>clipboard_export**. Den Aufruf packst du gleich noch mit dazu.



So sieht es bei mir aus.



Erstens Definition:

```
FORM write_to_clipboard USING
    value(it_customers) TYPE gtt_customers.*1
DATA: ld_rc TYPE i.
CALL METHOD cl_gui_frontend_services=>clipboard_export*2
IMPORTING
    data                = it_customers
CHANGING
    rc                  = ld_rc
EXCEPTIONS
    cntl_error          = 1
    error_no_gui        = 2
    not_supported_by_gui = 3
    OTHERS              = 4.
```

*1 Die Daten für die Kunden müssen natürlich über die Schnittstelle an das Unterprogramm übergeben werden.

*2 Der Methodenaufruf ist recht unspektakulär und erfordert nur zwei Parameterversorgungen: **data** und **rc**, wobei **rc** der **Rückgabecode** über Erfolg oder Misserfolg ist.

```

IF sy-subrc <> 0.*3
    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
        WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.
ENDFORM.

```

 Im Fall einer Ausnahme setzt die Methode **cl_gui_frontend_services=>clipboard_export** die **Systemvariablen** für die Nachricht. Fehlt dir etwas?

So auf Höhe Kommentar 2,5? Stimmt, das **flush** fehlt. Aus einem einfachen Grund: Es wird in der Methode **clipboard_export** aufgerufen. Das kannst du gerne kontrollieren, falls du es nicht glaubst!

Zweitens AUFRUF, der WHEN 'DOWN'-Abschnitt wächst:

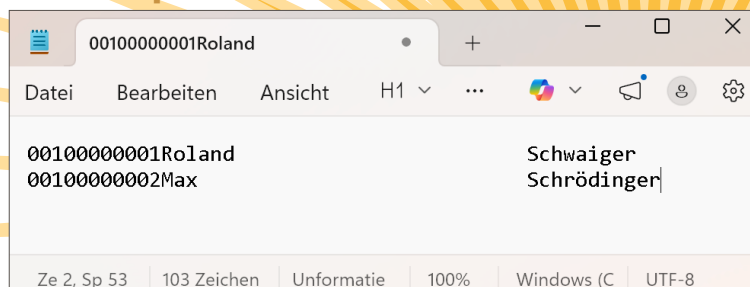
```

WHEN 'DOWN'.
* Die Kundendaten ermitteln
    PERFORM get_customer_data
        CHANGING
            gt_customers.
* Daten ins Clipboard schreiben
* Testen mit Strg + V im Notepad
    PERFORM write_to_clipboard
        USING
            gt_customers.

```

Den Aufruf hab ich direkt hinter das Ermitteln der Kundendaten gestellt. Und nun wird getestet. Als Vorbereitung öffne einen einfachen Texteditor. **Programm aktivieren, starten** und die Taste **Download** drücken. In den Editor wechseln und **Strg** + **V**.

Und was passiert?



Die Daten mit **Strg** + **V** im Texteditor eingefügt. Sehr praktisches Feature!

Download now!

Jetzt kommt das **Dateischreiben**. Auch nicht viel komplizierter als das, was wir bisher gemacht haben. Du musst nur wissen, dass die Methode **cl_gui_frontend_services=>gui_download** für den Download zuständig ist, den Rest erläutere ich noch.



[Schwierige Aufgabe]

Lege das Unterprogramm **customers_download** an, und implementiere darin den Aufruf der Methode **cl_gui_frontend_services=>gui_download**. Natürlich mit Aufruf des Unterprogramms im Abschnitt **DOWN**.

Definition DIE ERSTE:

***2** Der Aufruf von **gui_download** beeindruckt durch die Vielzahl an Formalparametern. **Unglaublich!** Ich werde nur ein paar wichtige herauspicken. Die anderen habe ich bereits gelöscht. Sie sind meinen Platzeinsparungsmaßnahmen zum Opfer gefallen. Der Parameter **filename** wird mit **id_file_path** versorgt, da in diesem der **absolute Pfad für die Datei** steckt.

Der **filetype** legt fest, in welchem Format die Daten geschrieben werden. Die Daten können zum Beispiel im **ASCII-Format** (Typ **'ASC'**), im **ASCII-Format mit IBM-Codepage** (Typ **'IBM'**), im **Spaltenformat** (Typ **'DAT'**) oder auch **binär** (Typ **'BIN'**) übertragen werden. Ich habe das Spaltenformat **'DAT'** gewählt, damit wir uns das dann im **Tabellenkalkulationsprogramm** ansehen können. **write_field_separator** fügt einen **Tabulator** zwischen den Spalten ein.

***1** Für den Download benötigen wir die Infos **WOHIN** (**id_file_path**) und **WAS** (**it_customers**).

Wird der Parameter **dat_mode** gesetzt, wird die Datei im **'DAT'**-Format gespeichert. Dabei werden **keine Konvertierungen** ausgeführt. Zahlen- und

Datumsfelder werden in einem Standardformat abgelegt, das den späteren Import durch **gui_upload** erlaubt. **confirm_overwrite** sorgt dafür, dass eine **Bestätigung** von dir angefordert wird, sofern die Datei **bereits existiert**. **show_transfer_status** regelt, ob die **Message** zum Transferstatus angezeigt wird oder nicht. Damit kannst du experimentieren, wenn du möchtest.

```
FORM customers_download USING id_file_path TYPE string
                           it_customers TYPE gtt_customers.*1
CALL METHOD cl_gui_frontend_services=>gui_download *2
EXPORTING
    filename                = id_file_path
    filetype                = 'DAT'
    write_field_separator   = abap_true
    dat_mode                = abap_true
    confirm_overwrite       = abap_true
*    show_transfer_status    = 'X'
CHANGING
    data_tab                = it_customers
EXCEPTIONS *3
    file_write_error        = 1
    no_batch                = 2
```

***3** Die Behandlung der Ausnahmen wird auch etwas Zeit in Anspruch nehmen. Wie du siehst, werden dir sehr feingranular mögliche Ausnahme-situationen geliefert.

gui_refuse_filetransfer	= 3
invalid_type	= 4
no_authority	= 5
unknown_error	= 6
header_not_allowed	= 7
separator_not_allowed	= 8
filesize_not_allowed	= 9
header_too_long	= 10
dp_error_create	= 11
dp_error_send	= 12
dp_error_write	= 13
unknown_dp_error	= 14
access_denied	= 15
dp_out_of_memory	= 16
disk_full	= 17
dp_timeout	= 18
file_not_found	= 19
dataprovder_exception	= 20
control_flush_error	= 21
not_supported_by_gui	= 22
error_no_gui	= 23
OTHERS	= 24



```

.
IF sy-subrc <> 0.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
    WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ELSE.
  MESSAGE 'Alle Kunden sind unten'(m01) TYPE 'I'. *4
ENDIF.
ENDFORM.

```

*4 Die Erfolgsnachricht

Die Implementierung des Aufrufs ist klar, oder? Hier der vollständige DOWNLOAD-Bereich:

```

  WHEN 'DOWN'.
* Die Kundendaten ermitteln
    PERFORM get_customer_data
      CHANGING
        gt_customers.
* Daten ins Clipboard schreiben

```

- * Zum Testen das Notepad oder irgendein anderes Programm
- * öffnen, in das du mit Strg + V einfügen kannst

```
PERFORM write_to_clipboard
      USING
          gt_customers.
```

- * Datei schreiben

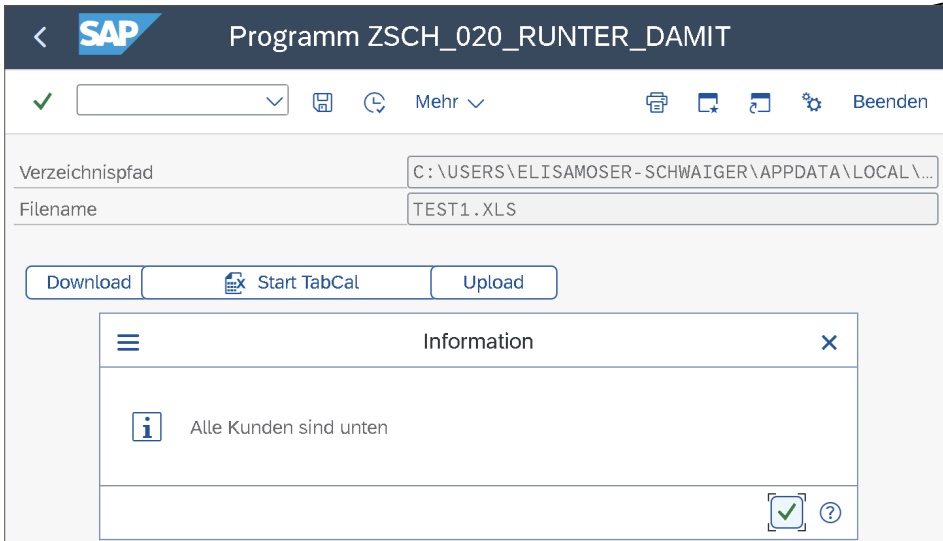
```
PERFORM customers_download USING
      gd_file_path
      gt_customers.
```

Ein kleiner Test zeigt uns das Folgende.

Apropos: Erwähnte ich, dass du vor dem Testen aktivieren musst?

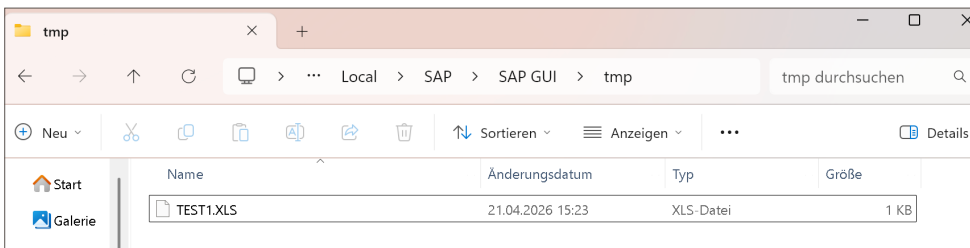
Nein, ich hab's aber schon erledigt. Bin schon länger fertig.

Autonome Zelle! Also dann: Testen des Programms und die Drucktaste **Download** drücken. **Und? Erfolgsnachricht?**



Alles wunderbar, die Daten sind unten.

Als skeptischer Mensch könnte man jetzt natürlich sagen: Super Erfolgsmeldung, aber sind die Daten wirklich geschrieben worden? Auf diese Frage war ich natürlich vorbereitet und kann auch sofort das nächste Bild präsentieren:



Das Ergebnis liegt im lokalen Dateiverzeichnis. Jetzt können wir ein wenig stolz sein.

Wenn du mutig bist, dann drückst du den **Download** noch mal.

Was passiert? **Kommt die Sicherheitsabfrage?**



[Einfache Aufgabe]

Woher kommt die Sicherheitsabfrage?

Die Datei gibt es schon, darum nachfragen!

*Natürlich vom Aufruf der Methode **gui_download**,
Parameter **confirm_overwrite**.*

Einen hamma noch – Upload

So, am Ende noch den **Upload**. Der hat aber wenig Neues für dich parat, da will ich dich jetzt nicht mit Fragen und Aufgaben plagen.

Frei-Haus-Lieferung der Implementierung des Unterprogramms **customers_upload**:

```
FORM customers_upload USING value(id_file_path) TYPE string
                          CHANGING ct_customers TYPE gtt_customers.
```

```
CALL METHOD cl_gui_frontend_services=>gui_upload *
```

EXPORTING

filename	=	id_file_path
filetype	=	'DAT'
has_field_separator	=	abap_true
dat_mode	=	abap_true

CHANGING

data_tab	=	ct_customers
----------	---	--------------

EXCEPTIONS

file_open_error	=	1
file_read_error	=	2
no_batch	=	3
gui_refuse_filetransfer	=	4
invalid_type	=	5
no_authority	=	6
unknown_error	=	7
bad_data_format	=	8

***1** Die **Parameter**, die wiederum nur eine Teilmenge der vorhandenen sind, decken sich bezüglich der Ausprägungen mit denen des **Downloads**.

```

header_not_allowed      = 9
separator_not_allowed   = 10
header_too_long         = 11
unknown_dp_error        = 12
access_denied           = 13
dp_out_of_memory        = 14
disk_full               = 15
dp_timeout              = 16
not_supported_by_gui    = 17
error_no_gui            = 18
OTHERS                  = 19

```

```
IF sy-subrc <> 0.
```

```

  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
    WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.

```

```
ELSE.
```

```
  MESSAGE 'Alle Kunden sind oben' TYPE 'I'.*2
```

```
ENDIF.
```

```
ENDFORM.
```

*2 Sogar die Erfolgsmeldung sieht ähnlich aus.

Der Aufruf steckt im Abschnitt **WHEN 'UP'**:

```
WHEN 'UP'.
```

```
* Datei holen
```

```

  PERFORM customers_upload USING
                                gd_file_path
                                CHANGING
                                gt_customers.

```

*1 Ich lade nur die Daten in die interne Tabelle **gt_customers**. Von hier weg könntest du übernehmen und zum Beispiel die Daten in einer Liste ausgeben.

So viel zum Thema Client-Up- und -Download.



[Einfache Aufgabe]

Du schon, ich nicht. Das sollte dir als weiteres Beispiel zum Üben dienen. Einfach die Methoden aus den Frontend Services verwenden. Wenn du spicken möchtest, findest du den vollständigen Code im Download-Material zum Buch.

Aber da fehlt doch noch der Start des Tabellenkalkulationsprogramms. Machen wir das nicht mehr?

Daten auf dem Applikationsserver

Auf dem Client bist du schon fit im Datenladen. Alle User können sich jetzt die Daten runter- und raufladen, wie es ihnen gefällt. Ab und zu benötigen wir aber einen einheitlichen Speicherort für Dateien, zum Beispiel für **Protokolle** oder **Austauschdaten**. Dazu bietet sich natürlich der **Applikationsserver** an. Wie sieht es aber auf dem Server aus?

Wie kannst du da Daten runter- und raufschieben?

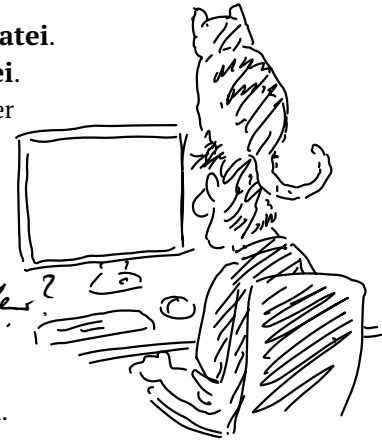
Zu diesem Behufe hat die SAP die **DATASET**-Operationen vorbereitet:

- ☛ **OPEN DATASET <file>** dient zum **Öffnen einer Datei** auf dem Applikationsserver.
- ☛ **CLOSE DATASET <file>** dient zum **Schließen der Datei**.
- ☛ **READ DATASET <file>** dient zum **Schreiben der Datei**.
- ☛ **TRANSFER <data> TO <file>** dient zum Schreiben der Daten in die **Datei**.

*Komisch: Nennt sich **READ** und bedeutet schreiben? Du hast schon wieder ein Kuckucksei gelegt, oder?*

Erwischt. Noch mal, und jetzt aber richtig:

- ☛ **READ DATASET <file>** dient zum **Auslesen der Datei**.



So, jetzt passt es!

Mit der Anweisung **OPEN DATASET** und dem Zusatz **FOR OUTPUT** kannst du schon mal eine **Datei öffnen**, um Inhalte

hineinzuschreiben. Falls du von der **Datei lesen** möchtest, dann verwendest du den Zusatz **FOR INPUT**. Und falls du **etwas anhängen** möchtest, steht dir der Zusatz **FOR APPENDING** zur Verfügung.

Die Art der **Codierung**, in der die Daten übertragen werden, lässt sich mit **IN BINARY MODE** oder **IN TEXT MODE** festlegen.

Ganz kurz und knackig könnte man das **OPEN DATASET** so schreiben:

```
OPEN DATASET file FOR access IN mode  
[position]  
[os_addition]  
[error_handling].
```

[Zettel]

Erstelle mit dem bisherigen Wissen den Befehl zum Öffnen der Datei **test.dat** für **Schreiben im Textmodus**. Lege dir dazu das ausführbare Programm **zsch_020_runter_damit_auf_as** an.

Eins ist klar: Die Datei muss vom aktuellen Applikationsserver aus erreichbar sein. Es ist mit **OPEN DATASET** nicht möglich, Dateien auf deinem Client zu bearbeiten.

*Kein Problem,
dafür hab ich eh die
cl_gui_frontend_services!*

Na eh, ich wollte es nur gesagt haben.

OPEN House – OPEN DATASET

Hast du das Beispiel schon fertig? Meine Lösung sieht so aus:

```
DATA: gd_file TYPE authb-filename VALUE `test.dat`,*1
      gd_result TYPE string.
DATA: gt_customers TYPE TABLE OF zsch017customer,*2
      gs_customer LIKE LINE OF gt_customers.
START-OF-SELECTION.
* Daten zu den Kunden lesen, ganz ungekapselt
  SELECT * FROM zsch017customer INTO TABLE gt_customers.*3
* Schreiben
  OPEN DATASET gd_file FOR OUTPUT IN TEXT MODE
                                ENCODING DEFAULT
                                WITH SMART LINEFEED.*4
```

***1** Zuerst benötige ich mal einen **Dateinamen**. Als Typ habe ich **auth-filename** verwendet, da dieser Typ zur Schnittstelle des Funktionsbausteins **AUTHORITY_CHECK_DATASET** passt, mit dem du prüfen kannst, ob jemand die Berechtigung hat, zu lesen, schreiben etc.

***2** Die Daten, die in die Datei zu schreiben sind, müssen auch irgendwo her kommen. Also lese ich sie von der DB in die interne Tabelle **gt_customer**.

***3** Iggitt, voll hässlich. Ein **SELECT**, das nicht verkapselt ist. Das könntest du bitte mithilfe eines Unterprogramms verbessern.

[Einfache Aufgabe]

Definiere das **read_customers**-Unterprogramm, in dem du die Daten liest.



***4** Jetzt wird es spannend. Ich öffne die Datei **gd_file** für die Ausgabe (**OUTPUT**) im Textmodus und lege fest, dass die Codierung der Daten mit **ENCODING DEFAULT** durchzuführen ist, und das bedeutet: In einem **Unicode-System** entspricht die Angabe einer Codierung in **UTF-8** und in einem **Nicht-Unicode-System** der Angabe **NON-UNICODE**. Mit dem Zusatz **WITH SMART LINEFEED** wird die **Zeilenende-Markierung** gemäß dem **Betriebssystem** des Applikationsservers festgelegt.

[Notiz]

Die Form von **Dateinamen** ist stark **betriebssystemabhängig**. Portable Programme kann man durch Verwendung des Funktionsbausteins **FILE_GET_NAME** erreichen. Dieser Funktionsbaustein liefert zu einem gegebenen **logischen Dateinamen** den zugehörigen **physischen Namen**.

Die für die plattformunabhängige Vergabe von Dateinamen benötigten Definitionen werden mit der Transaktion **FILE** mandantenübergreifend gepflegt. Mit der Transaktion SF07 kann eine Übersicht dieser Daten als Liste ausgegeben werden. Die Transaktion SF01 ermöglicht es, logische Dateinamen zusätzlich auch spezifisch für den aktuellen Mandanten zu definieren.

Nachdem die Datei nun geöffnet ist und die Daten vorhanden sind, kannst du die Daten hineinschreiben.

DatenTRANSFER – Daten schreiben

Wo liegt eigentlich die Datei, wenn sie geschrieben ist?

Ach ja, wo liegt sie denn? Das zeige ich dir, nachdem wir die Daten aus der Tabelle runtergeschrieben haben:

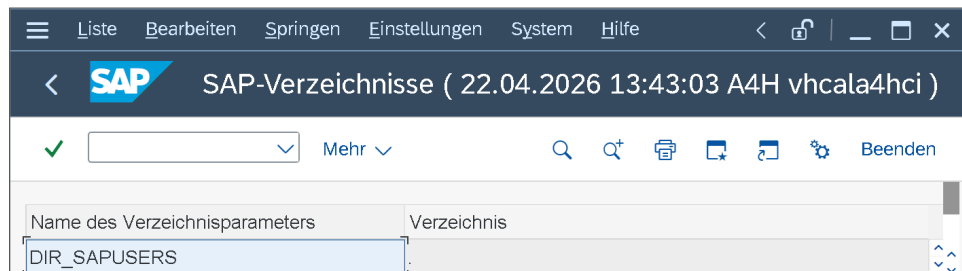
```
LOOP AT gt_customers INTO gs_customer.*1  
  TRANSFER gs_customer TO gd_file.*2  
ENDLOOP.  
CLOSE DATASET gd_file.*3
```

*1 Zeile für Zeile werden die Daten aus der internen Tabelle **gt_customers** der Kunden geholt.

*2 Mit der Anweisung **TRANSFER** wird der Inhalt des **Datenobjektes** in die **Datei** übertragen.

*3 **CLOSE DATASET** schließt die Datei.

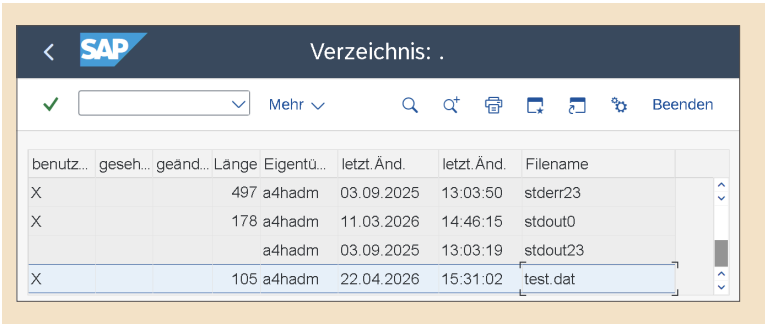
Fertisch! Jetzt kann ich dir die Ablage auf dem Applikationsserver zeigen. Mit der Transaktion **AL11** kannst du in das **Dateiverzeichnis** des **Applikations-servers** wechseln.



Die SAP-Verzeichnisse. Du findest die Datei im Verzeichnis . (Punkt). Wie die Bezeichnung lautet, musst du bei dir nachsehen, bei mir ist es **DIR_SAPUSERS**.

Per Doppelklick auf das Verzeichnis **DIR_SAPUSERS** gelangst du in dieses Verzeichnis, und dort findest du ganz unten die Datei **test.dat**.

Die ersten drei Spalten zeigen dir den **Status der Datei**, danach kommen **technische Informationen** und der **Name**.

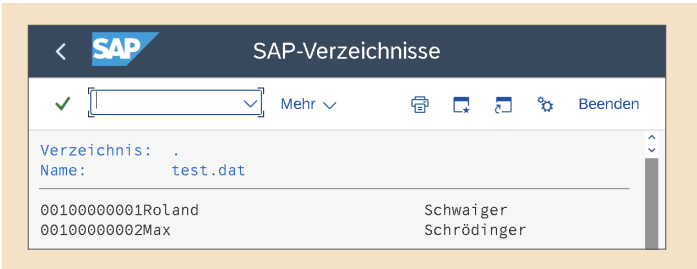


The screenshot shows the SAP file explorer interface. At the top, there's a header with the SAP logo and the title 'Verzeichnis: .'. Below the header is a search bar and a toolbar with icons for search, copy, paste, and other actions. The main area displays a table of files and directories. The table has columns for 'benutz...', 'geseh...', 'geänd...', 'Länge', 'Eigentü...', 'letzt. Änd.', 'letzt. Änd.', and 'Filename'. The 'test.dat' file is highlighted at the bottom of the list.

benutz...	geseh...	geänd...	Länge	Eigentü...	letzt. Änd.	letzt. Änd.	Filename
X			497	a4hadm	03.09.2025	13:03:50	stderr23
X			178	a4hadm	11.03.2026	14:46:15	stdout0
				a4hadm	03.09.2025	13:03:19	stdout23
X			105	a4hadm	22.04.2026	15:31:02	test.dat

Da sind ja einige Dateien drinnen! Unter anderem auch unsere Datei **test.dat**.

Weiter geht's auf unserer Reise zu den unentdeckten Details der Dateien. Noch ein Doppelklick auf die Datei **test.dat** zeigt dir dann die Inhalte.



The screenshot shows the SAP file explorer interface with the title 'SAP-Verzeichnisse'. Below the header, there's a search bar and a toolbar. The main area displays the contents of the 'test.dat' file. It shows a list of names and their corresponding IDs, with the last two names, 'Schwaiger' and 'Schrödinger', highlighted.

Verzeichnis: .	Name: test.dat
00100000001Roland	Schwaiger
00100000002Max	Schrödinger

Wow, alles drinnen, was vorher in der Datenbanktabelle war.



READ DATASET – Daten lesen

Genau. Natürlich geht es auch in die andere Richtung. Von der Datei ins Programm, und das nennt sich dann **READ DATASET**. Bevor du das Lesen beginnen kannst, musst du natürlich die Datei so öffnen, dass man daraus lesen kann.

```
OPEN DATASET gd_file FOR INPUT IN TEXT MODE
                                ENCODING DEFAULT
                                WITH SMART LINEFEED.*1

DO.*2
    READ DATASET gd_file INTO gd_result.*3
    IF sy-subrc <> 0.*4
        EXIT.
    ENDIF.
    WRITE / gd_result.*5
ENDDO.
CLOSE DATASET gd_file.*6
```

***1** Mit der Anweisung **OPEN DATASET** öffne ich die Datei zum Lesen (**FOR INPUT**), und zwar im **Textmodus**. Also genauso wie ich vorher die Daten geschrieben habe. Die anderen Einstellungen sind dir noch von vorhin bekannt.

***2** Mit einer **DO ... ENDDO**-Schleife kannst du so lange lesen gehen, bis keine Daten mehr vorhanden sind. Da musst du natürlich darauf achten, dass du die **Endlosschleife** auch wieder verlässt.

***3** Lesen! Von der Datei, und das Ergebnis wird in die Zielstruktur **gd_result** gestellt. Wenn du dir den Typ von **gd_result** nochmals ansiehst, wirst du feststellen ...

*Das ist ja ein string!
Runtergeschrieben
haben wir aber
strukturierte Daten vom
Typ zsch017address.*



***4** Falls beim Lesen der **sy-subrc <> 0** ist, bedeutet das, dass das **Dateiende** erreicht wurde. Zeit zum Aufhören, also **EXIT**.

Und den könnten wir uns natürlich wieder genauso holen. Einfach den Typ von **gd_result** verändern.

[Einfache Aufgabe]

Ändere den Typ von **gd_result** in den passenden Typ für eine strukturierte Datenübernahme.

***5** Und natürlich werden die Daten auch ausgegeben. In meinem Beispiel als Ganzes, da **gd_result** unstrukturiert ist.

*Wie als Ganzes? Im Ganzen
wie ein Brot oder
aufgeteilt in Schnitzel?*

gd_result ist vom Typ **string** und besitzt daher keine Komponenten.

***6** **CLOSE DATASET** schließt die Datei.

Damit bist du der Meister der Daten

außerhalb von Datenbanktabellen. Du könntest dir eine kleine Krone basteln und aufsetzen.

