Hash Objects - Reloaded

Arne Leißner
Entimo AG
Stralauer Platz 33-34
10243 Berlin
arne.leissner@entimo.de

Zusammenfassung

Der Beitrag "Geschwindigkeit ist nicht alles - wozu Hash Objekte wirklich gut sind" auf der 19. KSFE-Tagung demonstrierte mit anschaulichen Beispielen eine Vielfalt von Elementaraufgaben, die sich auch mit Hash Objekten lösen lassen. Der diesjährige Beitrag zeigt weitere Einsatzpotentiale auf, die sich insbesondere aus der Kombination dieser Techniken ergeben. Die Vorstellung dieser Ideen ist wieder mit einfachen und verständlichen Beispielen gewürzt. Gezeigt wird unter anderem, wie eine anspruchsvolle rekursive Verarbeitung wie das Auslesen eines Verzeichnisbaumes mit Hilfe eines Hash Objekts ganz unkompliziert und elegant verwirklicht werden kann. Selbstverständlich gibt es auch diesmal vorab wieder eine kurze Einführung in die Welt der Hash Objekte. Der Beitrag soll anregen, die spezifischen Vorteile von Hash Objekten gewinnbringend zu nutzen und effizienten Code zu schreiben.

Schlüsselwörter: Data Step, Hash-Objekt

1 Einführung

Mit dem Hash-Objekt steht dem Data-Step-Programmierer ein zusätzliches Datenobjekt zur Verfügung, das für die Ablage schlüsselbasierter Informationen im Hauptspeicher und deren sehr schnelles Wiederauffinden hervorragend geeignet ist.

Tabelle 1: Charakteristika von Hash-Objekten

Benefits
Zusätzliche dynamische Datenstruktur(en) im Direktzugriff des DATA Steps
Können bedingungsabhängig zur Execution Time erzeugt werden und sind flüchtig („leben“ nur für die Dauer eines Data Steps)
Einträge über Schlüsselwerte indiziert und auffindbar (Speicherposition = Bucket via Hashkey, Ablage erfolgt unsortiert)
Simple und composite Keys unterstützt
Jeder Eintrag kann ein Datentupel in vordefinierter Struktur aufnehmen (u.a. kann der als Referenz verwendete Schlüsselwert hier auch noch einmal explizit abgelegt werden)

Sehr schnelle Suchalgorithmen unabhängig von der Anzahl der Einträge (Hashfunktionen in Memory)
Können und dürfen groß werden (viele Einträge)
Freie Navigation (via Schlüssel od. sequentiell) → Direktadressierung → Multiadressierung → vorwärts / rückwärts mittels Iterator
Dynamische und agile Techniken für das anlegen, modifizieren und löschen von Einträgen (manuell und automatisch)
Verlinkung zu Data Step Variablen → Automatischer Informationsaustausch zwischen PDV und Hash-Objekt (wahlweise auch manuelle Zuweisungen)
Einfacher Import / Export von SAS Data Sets mit Data Set Optionen
Eingebaute Summenfunktion
Steuerung des Verhaltens via attribute tags
Objektorientierter Ansatz (Methoden, Attribute, Dot-Notation)

Die allein schlüsselabhängige Speicherposition und damit unsortierte Ablage der Einträge soll an nachfolgendem anschaulichen Beispiel kurz illustriert werden.

In einem kleinen Hotel gibt es aus früheren Zeiten noch einen alten Schlüsselkasten (unser Hash-Objekt) mit 11 durchnummerierten Haken für die Schlüssel der Gästezimmer. Das Hotel verfügt inzwischen bereits über 16 Zimmer mit den Zimmernummern:

1. Stockwerk: 11, 12, 14, 15, 16, 17
2. Stockwerk: 21, 22, 23, 24, 25, 26
3. Stockwerk: 31, 32, 33, 34

Mit einem Hash-Algorithmus sollen die 16 Schlüssel geeignet auf die 11 Haken verteilt werden, so dass für jeden Schlüssel die Hakenposition sehr schnell ermittelt werden kann. Ein sehr einfacher Hash-Algorithmus ist die Modulo-Funktion (Teilen mit Rest). Unter Verwendung des Teilers 11 ergeben sich für die obigen Zimmernummern folgende Hash-Werte:

Zimmer 11 → 0
Zimmer 12 → 1
Zimmer 14 → 3
Zimmer 15 → 4
Zimmer 16 → 5
Zimmer 17 → 6

Zimmer 21 → 10
Zimmer 22 → 0
Zimmer 23 → 1
Zimmer 24 → 2
Zimmer 25 → 3
Zimmer 26 → 4
Zimmer 31 → 9
Zimmer 32 → 10
Zimmer 33 → 0
Zimmer 34 → 1

Wird jetzt noch der Wert 1 addiert , so ist jedem Schlüssel eindeutig einer der Haken 1 bis 11 zugeordnet:

Haken 1 → Zimmerschlüssel 11, 22, 33
Haken 2 → Zimmerschlüssel 12, 23, 34
Haken 3 → Zimmerschlüssel 24
Haken 4 → Zimmerschlüssel 14, 25
Haken 5 → Zimmerschlüssel 15, 26
Haken 6 → Zimmerschlüssel 16
Haken 7 → Zimmerschlüssel 17
Haken 8
Haken 9
Haken 10 → Zimmerschlüssel 31
Haken 11 → Zimmerschlüssel 21, 32

Wird nach dem Schlüssel für das Zimmer 25 verlangt, steht unter erneuter Anwendung des Hash-Algorithmus fest, dass dieser auf Haken 4 hängt. Dort ist er (auch wenn sich dort noch der Schlüssel für das Zimmer 14 befindet) leicht und schnell zu finden.

Das Beispiel zeigt, dass anhängig von den konkreten Schlüsselwerten, der Größe des Hash-Objektes und des verwendeten Hash-Algorithmus einzelne Positionen des Hash-Objektes nicht, einfach oder mehrfach besetzt sein können. Das ist unabhängig davon, ob es mehr oder weniger Schlüsselausprägungen als Speicherpositionen im Hash-Objekt gibt. Gute Hash-Techniken zielen auf eine möglichst gleichmäßige Verteilung der Schlüsseleinträge.

2 Beispiele für Elementartechniken und Use Cases

Das Hash-Objekt ist nicht nur ein Turbo bei den Lookup-Technologien, sondern unterstützt eine Vielzahl von Anwendungsszenarien und erweitert die Menge der technischen Optionen zur Lösung ausgewählter Aufgabenstellungen.

Tabelle 2: Einsatzmöglichkeiten für das Hash-Objekt (Auswahl)

Techniken & Use Cases
Lookup
Sortierung
De-Duplizierung
Zählung
Summierung
Durchwanderung
Data Set Splitting
Mehrfachlesen
Fuzzy Merge
Top-<n>%
Gruppenstatistiken
Join
Rekursives Lookup
Relationsprüfungen
…

Für die in Tabelle 2 unterstrichenen Anwendungsszenarien finden sich in [2] einfache und aufeinander aufbauende Prinzipbeispiele (jeweils mit Aufgabenstellung, Lösungsidee, SAS-Programm). Sie sind so angelegt, dass keine ausführlichen Erklärungen zu dem Anliegen und den verwendeten Inputdaten erforderlich sind und dass die zur Illustration des Lösungsweges abgebildeten SAS-Programme in jedem Fall nur wenige Zeilen umfassen.

Allen diesen Beispielen ist gemein, dass Sie als One-Step-Samples konzipiert sind – die mehr oder weniger komplexe Arbeit mit Hash-Objekten erfolgt stets innerhalb der Grenzen eines einzelnen Data Steps. Eine Step-übergreifende Verarbeitung ist per Definition nicht vorgesehen und nicht unterstützt. Mit dem Ende eines Data Steps werden die darin angelegten Hash-Objekte (wie auch der PDV und ggf. LAG-Queues) automatisch aus dem Hauptspeicher gelöscht. Dabei kommt folgender Grundaufbau des Data Steps zur Anwendung:

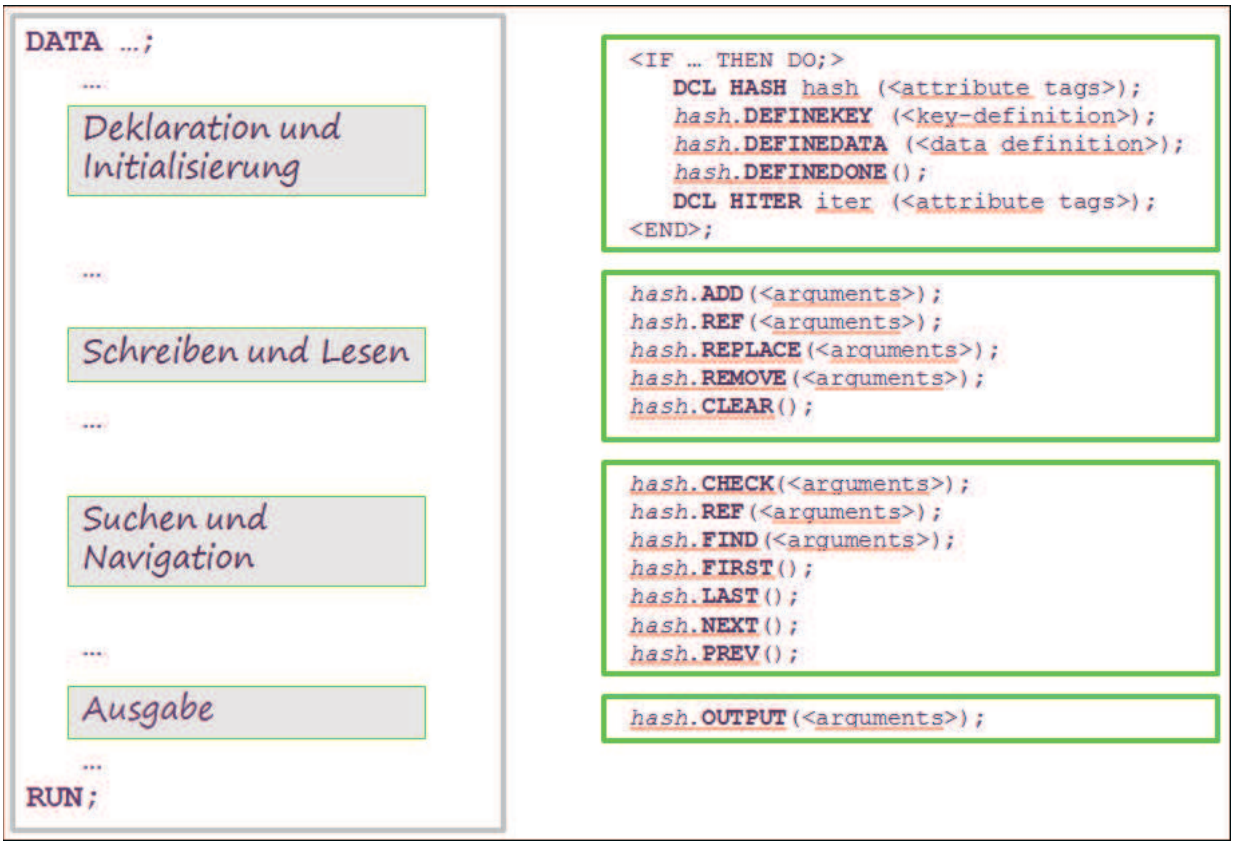


Abbildung 1: Prinzipaufbau eines Data Steps mit Nutzung Hash Objekt

In den nachfolgenden Abschnitten sollen zwei komplexere Aufgabenstellungen und deren Lösung unter Anwendung von Hash-Objekten vorgestellt und diskutiert werden. Verschiedene der oben aufgeführten Techniken werden dazu geeignet kombiniert.

3 Komplexbeispiele

3.1 Vergleich der Inhalte von Hash-Objekten

Aufgabe	Die Variablenstruktur und -eigenschaften einer SAS-Datei sollen gegen die in einer Metadatendatei hinterlegten Vorgaben geprüft werden. Anwendungsfall: Patientendatei AE mit Adverse Events aus einer klinischen Studie mit den zugehörigen Metadaten in SAS-Datei META_AE. Zu beachten: Nicht alle in den Metadaten definierten Attribute sind mandatory. Sie können somit in der AE-Patientendatei fehlen.

Lösungsidee	1. Die Metadatendatei und die abgeleiteten Strukturinformationen der Datendatei in zwei Hash-Objekte einlesen. 2. Prüfen, ob Hash-Objekte in Inhalt und Struktur identisch sind → Vollständige Übereinstimmung Patientendatenstruktur und Metadaten 3. In der Patientendatei nicht enthaltene optionale Spalten aus der Betrachtung ausschließen (aus dem Hash-Objekt mit den eingelesenen Metadaten entfernen) und erneut prüfen. 4. Abweichungen identifizieren (abweichende Spaltenanzahl, unterschiedliche Spalten, unterschiedliche Attributeigenschaften)
Verwendete Features	attribute-tag: DATASET Data Set Optionen Methoden: EQUALS() FIRST() NEXT() FIND() REMOVE() REPLACE() Attribut: NUM_ITEMS Hash Iterator

3.1.1 Die Eingangsdaten

Die Metadatendatei META_AE definiert 25 Variablen, davon ist nur eine (AESPID) als optional deklariert und kann somit in der Patientendatei fehlen. Für die Aufgabenstellung sind nur die in den Spalten „name“, „label“, „type“, „length“, „seq“ und „format“ definierten Informationen relevant, da diese mit den von SAS bereitgestellten Strukturinformationen für die Patientendatei vergleichbar sind. Weiterhin wird für die Steuerung der Prüfung die in der Variablen „mand(atory)“ hinterlegte Information benötigt.

	name	label	type	length	seq	key	format	formatl	formatd	mand	null	intkey	docvar	de
1	STUDYID	Study Identifier	c	12	1	1		.	.	Yes	Yes	No	.	
2	USUBJID	Unique Subject Identifier	c	11	2	2		.	.	Yes	Yes	No	.	
3	AETERM	Reported Term for the Adverse Event	c	13	3	.		.	.	Yes	Yes	No	.	
4	AESTDTC	Start Date/Time of Adverse Event	c	10	4	.		.	.	Yes	Yes	No	.	
5	AESEQ	Sequence Number	n	8	5	4		.	.	Yes	Yes	No	.	
6	DOMAIN	Domain Abbreviation	c	2	6	3		.	.	Yes	Yes	No	.	
7	AESPID	Sponsor-Defined Identifier	c	3	7	.		.	.	No	Yes	No	.	
8	AEDECOD	Dictionary-Derived Term	c	46	8	.		.	.	Yes	Yes	No	.	
9	AEBODSYS	Body System or Organ Class	c	67	9	.		.	.	Yes	Yes	No	.	
10	AESEV	Severity/Intensity	c	8	10	.		.	.	Yes	Yes	No	.	
11	AESER	Serious Event	c	1	11	.		.	.	Yes	Yes	No	.	
12	AEREL	Causality	c	8	12	.		.	.	Yes	Yes	No	.	
13	AEOUT	Outcome of Adverse Event	c	12	13	.		.	.	Yes	Yes	No	.	
14	AESCAN	Involves Cancer	c	1	14	.		.	.	Yes	Yes	No	.	
15	AESCONG	Congenital Anomaly or Birth Defect	c	1	15	.		.	.	Yes	Yes	No	.	
16	AESDISAB	Persist or Signif Disability/Incapacity	c	1	16	.		.	.	Yes	Yes	No	.	
17	AESDTH	Results in Death	c	1	17	.		.	.	Yes	Yes	No	.	
18	AESHOSP	Requires or Prolongs Hospitalization	c	1	18	.		.	.	Yes	Yes	No	.	
19	AESLIFE	Is Life Threatening	c	1	19	.		.	.	Yes	Yes	No	.	
20	AESOD	Occurred with Overdose	c	1	20	.		.	.	Yes	Yes	No	.	
21	AEENDTC	End Date/Time of Adverse Event	c	10	21	.		.	.	Yes	Yes	No	.	
22	AESTDY	Study Day of Start of Adverse Event	n	8	22	.		.	.	Yes	Yes	No	.	
23	AEENDY	Study Day of End of Adverse Event	n	8	23	.		.	.	Yes	Yes	No	.	
24	AEDTC	Date/Time of Collection of Adverse Event	c	10	24	.		.	.	Yes	Yes	No	.	
25	AEACN	Action Taken with Study Treatment	c	30	25	.		.	.	Yes	Yes	No	.	

Abbildung 2: Metadatendatei META_AE

Die Patientendatei umfasst 23 Variablen (DOMAIN an Position 6 und AESPID an Position 7 sind nicht enthalten).

	STUDYID	USUBJID	AETERM	AESTDTC	AESEQ	AEDECOD	AEBODSYS	AESEV
1	CDISCPILOT01	01-701-1015	VERBATIM_0995	2014-01-03	1	APPLICATION SITE ERYTHEMA	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MILD
2	CDISCPILOT01	01-701-1015	VERBATIM_1126	2014-01-09	3	DIARRHOEA	GASTROINTESTINAL DISORDERS	MILD
3	CDISCPILOT01	01-701-1015	VERBATIM_1219	2014-01-03	2	APPLICATION SITE PRURITUS	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MILD
4	CDISCPILOT01	01-701-1023	VERBATIM_0300	2012-08-07	1	ERYTHEMA	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MILD
5	CDISCPILOT01	01-701-1023	VERBATIM_0300	2012-08-07	4	ERYTHEMA	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MILD
6	CDISCPILOT01	01-701-1023	VERBATIM_1549	2012-08-07	2	ERYTHEMA	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MODERAT
7	CDISCPILOT01	01-701-1023	VERBATIM_1650	2012-08-26	3	ATRIOVENTRICULAR BLOCK SECOND DEGREE	CARDIAC DISORDERS	MILD
8	CDISCPILOT01	01-701-1028	VERBATIM_0578	2013-08-08	2	APPLICATION SITE PRURITUS	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MILD
9	CDISCPILOT01	01-701-1028	VERBATIM_1157	2013-07-21	1	APPLICATION SITE ERYTHEMA	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MILD
10	CDISCPILOT01	01-701-1034	VERBATIM_0555	2014-11-02	2	FATIGUE	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MILD
11	CDISCPILOT01	01-701-1034	VERBATIM_1219	2014-08-27	1	APPLICATION SITE PRURITUS	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MILD
12	CDISCPILOT01	01-701-1047	VERBATIM_0130	2013-02-12	1	HIATUS HERNIA	GASTROINTESTINAL DISORDERS	MODERAT
13	CDISCPILOT01	01-701-1047	VERBATIM_0130	2013-02-12	2	HIATUS HERNIA	GASTROINTESTINAL DISORDERS	MODERAT
14	CDISCPILOT01	01-701-1047	VERBATIM_0197	2013-03-10	4	BUNDLE BRANCH BLOCK LEFT	CARDIAC DISORDERS	MILD
15	CDISCPILOT01	01-701-1047	VERBATIM_0579	2013-03-06	3	UPPER RESPIRATORY TRACT INFECTION	INFECTIONS AND INFESTATIONS	MILD
16	CDISCPILOT01	01-701-1097	VERBATIM_0300	2014-01-03	1	ERYTHEMA	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MILD
17	CDISCPILOT01	01-701-1097	VERBATIM_0758	2014-03-21	5	PRURITUS GENERALISED	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MODERAT
18	CDISCPILOT01	01-701-1097	VERBATIM_0758	2014-04-19	7	PRURITUS GENERALISED	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MODERAT
19	CDISCPILOT01	01-701-1097	VERBATIM_0990	2014-02-20	2	PRURITUS GENERALISED	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MODERAT
20	CDISCPILOT01	01-701-1097	VERBATIM_0990	2014-03-31	6	PRURITUS GENERALISED	SKIN AND SUBCUTANEOUS TISSUE DISORDERS	MODERAT
21	CDISCPILOT01	01-701-1097	VERBATIM_1219	2014-02-21	4	APPLICATION SITE PRURITUS	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MILD
22	CDISCPILOT01	01-701-1097	VERBATIM_1219	2014-02-21	10	APPLICATION SITE PRURITUS	GENERAL DISORDERS AND ADMINISTRATION SITE CONDITIONS	MODERAT
23	CDISCPILOT01	01-701-1097	VERBATIM_1230	2014-04-19	8	PHARYNGOLARYNGEAL PAIN	RESPIRATORY, THORACIC AND MEDIASTINAL DISORDERS	MILD
24	CDISCPILOT01	01-701-1097	VERBATIM_1522	2014-02-20	3	APPLICATION SITE VESICLES	GENERAL DISORDERS AND	MILD

Abbildung 3: Patientendatei AE

Die Strukturinformationen zu der Patientendatei finden sich in der Dictionary Table SASHELP.VCOLUMN. Für die Aufgabenstellung sind die in den Spalten „name“,

„type“, „length“, „varnum“, „label“ und „format“ definierten Informationen relevant, da diese mit den Strukturvorgaben aus den Metadaten vergleichbar sind.

	libname	memna	memtype	name	type	length	npos	varnum	label	format	info
1	D_LIB	AE	DATA	STUDYID	char	12	24	1	Study Identifier		
2	D_LIB	AE	DATA	USUBJID	char	11	36	2	Unique Subject Identifier		
3	D_LIB	AE	DATA	AETERM	char	13	47	3	Reported Term for the Adverse Event		
4	D_LIB	AE	DATA	AESTDTC	char	10	60	4	Start Date/Time of Adverse Event		
5	D_LIB	AE	DATA	AESEQ	num	8	0	5	Sequence Number		
6	D_LIB	AE	DATA	AEDECOD	char	46	70	6	Dictionary-Derived Term		
7	D_LIB	AE	DATA	AEBODSYS	char	67	116	7	Body System or Organ Class		
8	D_LIB	AE	DATA	AESEV	char	8	183	8	Severity/Intensity		
9	D_LIB	AE	DATA	AESER	char	1	191	9	Serious Event		
10	D_LIB	AE	DATA	AEREL	char	8	192	10	Causality		
11	D_LIB	AE	DATA	AEOUT	char	12	200	11	Outcome of Adverse Event		
12	D_LIB	AE	DATA	AESCAN	char	1	212	12	Involves Cancer		
13	D_LIB	AE	DATA	AESCONG	char	1	213	13	Congenital Anomaly or Birth Defect		
14	D_LIB	AE	DATA	AESDISAB	char	1	214	14	Persist or Signif Disability/Incapacity		
15	D_LIB	AE	DATA	AESDTH	char	1	215	15	Results in Death		
16	D_LIB	AE	DATA	AESHOSP	char	1	216	16	Requires or Prolongs Hospitalization		
17	D_LIB	AE	DATA	AESLIFE	char	1	217	17	Is Life Threatening		
18	D_LIB	AE	DATA	AESOD	char	1	218	18	Occurred with Overdose		
19	D_LIB	AE	DATA	AEENDTC	char	10	219	19	End Date/Time of Adverse Event		
20	D_LIB	AE	DATA	AESTDY	num	8	8	20	Study Day of Start of Adverse Event		
21	D_LIB	AE	DATA	AEENDY	num	8	16	21	Study Day of End of Adverse Event		
22	D_LIB	AE	DATA	AEDTC	char	10	229	22	Date/Time of Collection of Adverse Event		
23	D_LIB	AE	DATA	AEACN	char	30	239	23	Action Taken with Study Treatment		

Abbildung 4: SASHELP.VCOLUMN für AE

3.1.2 Der Programmablauf

Programmabschnitt 1: Initialisierung

```
data _null_;
   /* 0 Initialisierung */
   /* 0.1 Variablen deklarieren */
   IF 0 THEN SET m_lib.meta_ae;

   /* 0.2 Metadaten in Hash Object einlesen */
   DCL HASH h_meta (DATASET: "m_lib.meta_ae");
   h_meta.DEFINEKEY("name");
   h_meta.DEFINEDATA("name", "label", "type", "length", "seq", "format");
   h_meta.DEFINEDONE();

   /* 0.3 Strukturinfos der Datendatei via Dictionary Table einlesen" */
   DCL HASH h_data (DATASET: "sashelp.vcolumn(WHERE=(libname = 'D_LIB'
                    and memname = 'AE') RENAME = (varnum = seq))");
   h_data.DEFINEKEY("name");
   h_data.DEFINEDATA("name", "label", "type", "length", "seq", "format");
   h_data.DEFINEDONE();
...

NOTE: There were 25 observations read from the data set M_LIB.META_AE.
NOTE: There were 23 observations read from the data set SASHELP.VCOLUMN.
      WHERE (libname='D_LIB') and (memname='AE');
```

Es werden zwei Hash-Objekte deklariert und unter Verwendung das DATASETS-Tags initial mit den relevanten Strukturinformationen gefüllt. Das Hash-Objekt *h_meta* liest direkt aus der Metadatendatei, das Hash-Objekt *h_data* nimmt aus der Dictionary Table SASHELP.VCOLUMN unter Verwendung von Data Set Optionen Strukturinformationen zu der Patientendatei auf. Beide Hash-Objekte haben anschließend eine identisch definierte Struktur.

Als Schlüsselname fungiert der Attributname. Dieser wird zusammen mit den Angaben zu „label“, „type“, „length“, „seq“ und „format“ als Datentupel im Hash-Objekt abgelegt. Schlüsselvariablen und Eintragsdaten müssen Data Step Variablen und somit im PDV deklariert sein. Die vorangestellte SET-Anweisung übernimmt diese Aufgabe, ohne (wegen IF 0 …) Sätze einzulesen.

Soll der Inhalt der Hash-Objekte überprüft werden, kann dies am einfachsten durch die Ausgabe in eine SAS-Datei erfolgen:

```
...
   h_data.OUTPUT(DATASET "work.data");
   h_meta.OUTPUT(DATASET "work.meta");
...

NOTE: The data set WORK.DATA has 23 observations and 6 variables.
NOTE: The data set WORK.META has 25 observations and 6 variables.
```

Programmabschnitt 2: Identitätsvergleich

```
...
   /* 1.1 Vollständige Übereinstimmung mit deklarierter Struktur */
   h_meta.EQUALS(HASH: "h_data", RESULT: eq);
   IF eq THEN PUT "Alles OK, alle deklarierten Variablen in Datendatei";
...
```

Die EQUALS-Methode Funktion prüft, ob zwei Hash-Objekte in Inhalt und Struktur identisch sind. Herangezogen werden dabei folgende Kriterien:

1. Die Speichergröße im Hauptspeicher
2. Die Zahl der Einträge
3. Die definierten Strukturen bzgl. Schlüssel- und Datenvariablen
4. Die Schlüssel- und Datenwerte an den entsprechenden Positionen

Das Prüfergebnis wird in eine frei wählbare Data Step Variable zurückgeliefert. Der Wert „1“ signalisiert Identität und wäre in dem Beispiel so zu interpretieren, dass alle deklarierten Variablen (unabhängig davon ob mandatory oder nicht) auch in der Patientendatei enthalten sind. In der Beispielpatientendatei fehlt jedoch u.a. die optionale Variable AESPID.

Programmabschnitt 3: Nicht vorhandene optionale Variablen aus Prüfung eliminieren

```
...
   /* 1.2 Übereinstimmung, falls einzelne opt. Var. nicht in Datendatei */
   IF NOT eq THEN DO;
      /* 1.2.1 Optionale Attribute in weiteres Hash Object einlesen */
      DCL HASH h_opt (DATASET: "m_lib.meta_ae(WHERE = (mand = 'N')
                               KEEP = name mand seq)");
      h_opt.DEFINEKEY("name");
      h_opt.DEFINEDATA("name", "seq");
      h_opt.DEFINEDONE();

      /* 1.2.2 Iteratoren definieren */
      DCL HITER h_opt_iter("h_opt");
```

```
      /* 1.2.3 Metadaten um nicht vorhandene opt. Spalten reduzieren */
      IF h_opt.NUM_ITEMS > 0 THEN DO;
         rc = h_opt_iter.FIRST();
         DO WHILE (rc = 0);
            IF h_data.FIND() THEN DO;
               h_meta.REMOVE();
            END;
            rc = h_opt_iter.NEXT();
         END;
         h_meta.EQUALS(HASH: "h_data", RESULT: eq);
         IF eq THEN PUT "Alles OK, einige opt. Var. nicht in Datendatei";
      END;
   END;
...
```

Es wird bedingungsabhängig, also zur Execution Time, ein neues Hash-Objekt *h_opt* angelegt und mit Informationen (Name und Position) zu den optionalen Variablen gefüllt. Es soll dann für jede optionale Variable geprüft werden, ob diese in der Datendatei existiert (d.h. ob in dem Dash-Objekt *h_data* ein entsprechender Eintrag gefunden wird). Wenn nicht, soll der Eintrag aus dem Hash-Objekt *h_meta* entfernt werden, um ihn von den weiteren Betrachtungen auszuschließen.

Um die Liste der optionalen Variablen (das Hash-Objekt *h_opt*) „durchwandern“ zu können wird ein zusätzliches Iterator-Objekt benötigt, in unserem Fall *h_opt_iter*. Mit den Methoden FIRST() und NEXT() lässt sich dieser Durchlauf programmtechnisch organisieren. Die Positionierung auf einem Item liefert automatisch (da auch als Datenattribut definiert) den Schlüsselwert in die Data Step Variable „name“ zurück. Dieser Wert der Variablen „name“ wird seinerseits verwendet, um mittels FIND-Methode in dem Hash-Objekt *h_data* nachzuschauen, ob die aktuell betrachtete Variable in der Datendatei vorhanden ist. Wenn nicht, wird sie mit REMOVE aus der Liste der Metadateneinträge entfernt.

In unserem Beispiel ist nur die Variable AESPID als optional deklariert. Das neue Hash-Objekt *h_opt* beinhaltet somit nur genau einen Eintrag. Da der Schlüssel AESPID nicht in dem Hash-Objekt *h_data* gefunden wird, wird er aus *h_meta* als nicht relevant entfernt. Es verbleiben dort 24 Einträge.

Die anschließende erneute Ausführung der EQUALS-Methode wird jedoch stets ein verfälschtes Ergebnis liefern, da die Sequence-Nummern für den Vergleich nicht angepasst wurden. Fehlt die an Position 7 erwartete optionale Variable AESPID in den Patientendaten, rutschen alle nachfolgenden Variablen um eine Position nach oben. Die Metadateninformationen müssen also zunächst entsprechend der neuen Situation angepasst werden. Der oben beschriebene Programmabschnitt ist zu erweitern.

```
...
   /* 1.2.2 Iteratoren definieren */
   ...
      DCL HITER h_meta_iter("h_meta");

      /* 1.2.3 Metadaten um nicht vorhandene opt. Spalten reduzieren */
      IF h_opt.NUM_ITEMS > 0 THEN DO;
         ...
```

```
            IF h_data.FIND() THEN DO;
               h_meta.REMOVE();
               /* SEQ-Nummern anpassen */
               rem_seq = seq;
               rc = h_meta_iter.FIRST();
               DO WHILE (rc = 0);
                  IF seq > rem_seq THEN DO;
                     seq = seq - 1;
                     rc = h_meta.REPLACE();
                  END;
                  rc = h_meta_iter.NEXT();
               END;
...
```

Mit Hilfe eines weiteren Iterators *h_meta_iter* wird nach dem Löschen eines Eintrages für alle verbliebenen Einträge in *h_meta* geprüft, ob Einträge mit einer höheren Sequence-Nummer als die des gelöschten Objekts (in der Variablen rem_seq gemerkt) vorliegen. Wenn ja, werden diese um den Wert 1 reduziert und in dem Hash-Objekt *h_meta* mittels REPLACE-Methode ersetzt.

Im konkreten Beispiel werden alle SEQ-Werte größer oder gleich 8 wegen der fehlenden Variablen 7 (AESPID) modifiziert.

Programmabschnitt 4: Abweichungen im Detail

```
...
   /* 1.3 Abweichungen im Detail */
   IF NOT eq THEN DO;
      /* 1.3.1 Anzahl der Variablen unterschiedlich*/
      IF h_meta.NUM_ITEMS ne h_data.NUM_ITEMS THEN
         PUT "Fehler: Variablenanzahl unterschiedlich";
...
```

Zunächst wird geprüft, ob die Anzahl der in den Metadaten definierten Attribute mit der Anzahl der in der Patientendatei tatsächlich vorhandenen Attribute übereinstimmt (nach Bereinigung bzgl. der nicht vorhandenen optionalen Variablen).

Das Attribut NUM_ITEMS beinhaltet die Zahl der Einträge eines hash-Objekts. Es kann für die beiden Hash-Objekte *h_meta* und *h_data* direkt verglichen werden.

```
...
      /* 1.3.2 Variablen aus Metadaten nicht in Datendatei */
      rc = h_meta_iter.FIRST();
      DO WHILE (rc = 0);
         IF h_data.FIND() THEN not_in_data = 1;
         rc = h_meta_iter.NEXT();
      END;
      IF not_in_data THEN PUT "Fehler: Variable(n) fehlen in Datendatei";
...
```

Unter Anwendung der schon mehrfach benutzten Iterator-Methoden FIRST() und NEXT() werden nacheinander die Einträge aus den Metadaten geprüft, ob die entsprechenden Spalten auch in der Datendatei vorhanden sind.

```
...
      /* 1.3.3 Variablen aus Datendatei nicht in Metadaten */
      DCL HITER h_data_iter("h_data");

      rc = h_data_iter.FIRST();
      DO WHILE (rc = 0);
         IF h_meta.FIND() THEN not_in_metadata = 1;
         rc = h_data_iter.NEXT();
      END;
      IF not_in_meta THEN PUT "Fehler: Undeklarierte Variable(n)";
...
```

Die Prüfung, ob in der Datendatei gefundene Spalten auch in den Metadaten deklariert sind, erfolgt in ähnlicher Art und Weise unter Verwendung eines weiteren Hash-Iterators *h_iter_data*.

```
...
      /* 1.3.4 Var.infos unterschiedlich (Typ|Länge|Label|Format|Seq) */
      rc = h_meta_iter.FIRST();
      DO WHILE (rc = 0);
         type_m = type; length_m = length; label_m = label;
         format_m = format; seq_m = seq;
         IF NOT h_data.FIND() THEN DO;
            if type ne type_m or length ne length_m or label ne label_m or
               format ne format_m or seq ne seq_m THEN diff_values = 1;
         END;
         rc = h_meta_iter.NEXT();
      END;
      IF diff_values THEN PUT "Fehler: Unterschiede in den Properties";
   END;
RUN;
```

Für Variablen, die sowohl in den Metadaten deklariert, als auch in den Daten tatsächlich vorhanden sind (die Schnittmenge), wird abschließend geprüft, ob die weiteren Eigenschaften bzgl. Typ, Länge, Label, Format und Position in der Datei übereinstimmen. Hierbei handelt es sich um eine wiederholte Anwendung der Methoden FIRST(), NEXT() und FIND(). Da beide Hash-Objekte ihre Datenwerte an die gleichen Data Step Variablen „type“, „length“, „label“, „format“, und „seq“, übergeben, müssen die Informationen aus dem einem Hash-Objekt zunächst zwischengespeichert werden (gleichnamige Data Step Variablen mit dem Suffix „_m“), um sie anschließend mit den Informationen aus dem anderen Hash-Objekt zu vergleichen.

3.1.3 Erweiterungsvorschläge

Das Programmbeispiel aus dem vorangegangenen Abschnitt stellt lediglich fest, ob es strukturelle Übereinstimmung oder Abweichungen einer bestimmten Art ermittelt hat. Darauf aufbauend lassen sich folgende Erweiterungen realisieren:

- Protokollierung des Strukturvergleiches im Detail.
- Ergänzung um inhaltliche Konsistenzprüfungen.

3.2 Rekursive Verarbeitung mit dynamischem Hash-Objekt

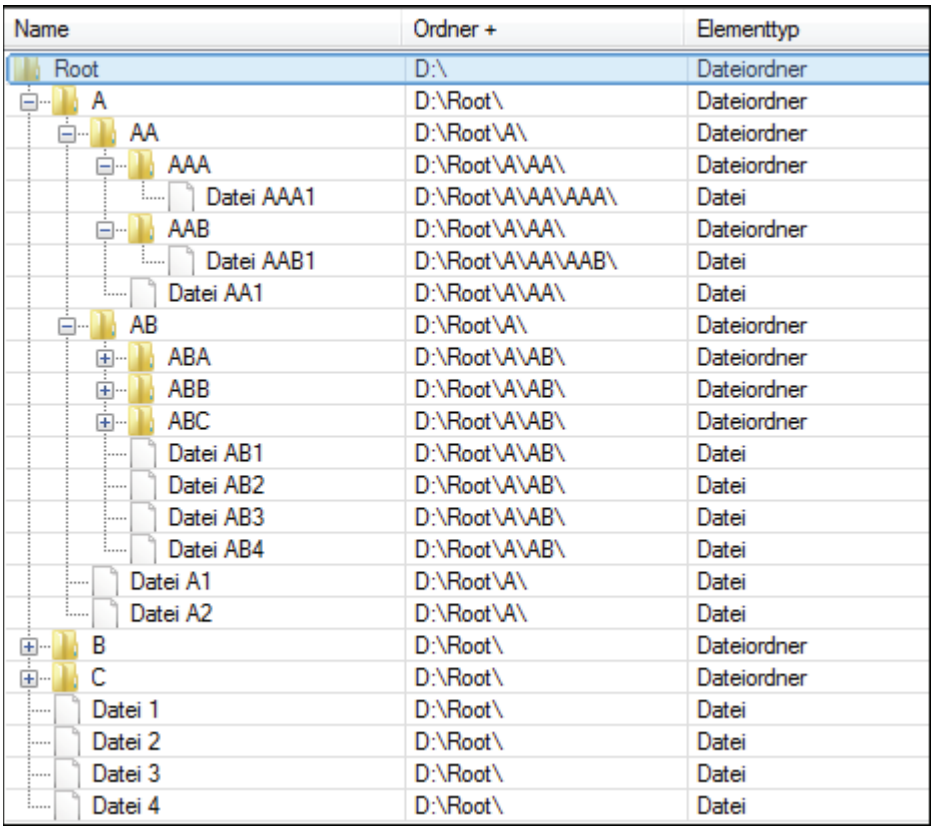
Aufgabe	Es sollen rekursiv (unter Einbeziehung aller Unterverzeichnisse) aber ohne rekursive Programmierung alle Dateien aus einem Verzeichnisbaum ermittelt werden.
Lösungsidee	1. Hash-Objekt zur Verwaltung von Verzeichnispfaden anlegen und mit dem Root-Pfad als zunächst einzigem Eintrag füllen. 2. Schleife über das Hash-Objekt ausführen: Solange noch Einträge (Verzeichnispfade) vorhanden sind: - den letzten Verzeichniseintrag auswählen - alle in dem Verzeichnis gefundenen Dateien in SAS-Datei schreiben - alle in dem Verzeichnis gefundenen Verzeichnisse (Unterverzeichnisse) dem Hash-Objekt hinzufügen - den aktuell bearbeiteten Verzeichniseintrag im Hash-Objekt löschen
Verwendete Features	attribute-tag: ORDERED Methoden: FIRST() LAST() ADD() REMOVE() Attribut: NUM_ITEMS Hash Iterator

3.2.1 Die Eingangssituation

Das Startverzeichnis „Root“ und seine mehrstufigen Unterverzeichnisse beinhalten verschiedene Dateien, die zu ermitteln sind.

Hierzu könnte eine Funktion programmiert werden, die für genau ein aktuell ausgewähltes Verzeichnis alle darin enthaltenen Dateien ermittelt, die Existenz von Unterverzeichnissen bestimmt und sich selbst (rekursiv) für das nächste noch nicht bearbeitet Unterverzeichnis aufruft. Anspruchsvoll wird eine derartige Programmierung insbesondere dadurch, dass nach dem Erreichen der tiefsten Verzeichnisstufe eines Zweiges (z.B. nach Analyse des Verzeichnisses AAA) die Rekursion an dieser Stelle abgebrochen und wieder eine Bottom-Up-Navigation eingeleitet werden muss. Auf diesem Wege müssen noch nicht untersuchte Verzeichnisse auf höherer Stufe (z.B. das zu AAA parallele Verzeichnis AAB) analysiert werden, bevor ein ggf. weiteres vorhandenes Root-Unterverzeichnis ausgewertet wird.

Gesucht wird ein effizienter linearer Algorithmus unter Verwendung eines Hash-Objektes. Im gewählten Beispiel befinden sich 25 Dateien in 19 Ordnern.



Name	Ordner +	Elementtyp
Root	D:\	Dateiordner
A	D:\Root\	Dateiordner
AA	D:\Root\A\	Dateiordner
AAA	D:\Root\A\AA\	Dateiordner
Datei AAA1	D:\Root\A\AA\AAA\	Datei
AAB	D:\Root\A\AA\	Dateiordner
Datei AAB1	D:\Root\A\AA\AAB\	Datei
Datei AA1	D:\Root\A\AA\	Datei
AB	D:\Root\A\	Dateiordner
ABA	D:\Root\A\AB\	Dateiordner
ABB	D:\Root\A\AB\	Dateiordner
ABC	D:\Root\A\AB\	Dateiordner
Datei AB1	D:\Root\A\AB\	Datei
Datei AB2	D:\Root\A\AB\	Datei
Datei AB3	D:\Root\A\AB\	Datei
Datei AB4	D:\Root\A\AB\	Datei
Datei A1	D:\Root\A\	Datei
Datei A2	D:\Root\A\	Datei
B	D:\Root\	Dateiordner
C	D:\Root\	Dateiordner
Datei 1	D:\Root\	Datei
Datei 2	D:\Root\	Datei
Datei 3	D:\Root\	Datei
Datei 4	D:\Root\	Datei

Abbildung 5: Tree View des auszuwertenden Verzeichnisbaum

3.2.2 Der Programmablauf

Programmabschnitt 1: Initialisierung

```
%LET root_dir = d:\root;

data work.files(keep = path member);
   /* 1. Initialisierung */
   /* 1.1 Hash-Objekt und Iterator anlegen */
   LENGTH no_dir 8 path $ 500;

   DCL HASH h_dir(ORDERED: "A");
   h_dir.DEFINEKEY("no_dir");
   h_dir.DEFINEDATA("no_dir", "path");
   h_dir.DEFINEDONE();
   DCL HITER h_dir_iter("h_dir");

   /* 1.2 Startverzeichnis setzen*/
   path = "&root_dir";
   anz_dir + 1;
   no_dir = 1;
   h_dir.ADD();
...
```

Es wird ein Hash-Objekt definiert, in das nacheinander und dynamisch alle gefundenen und noch zu verarbeitenden Unterverzeichnisse aufgenommen werden können. Jedes dieser Verzeichnisse erhält eine fortlaufende Nummer als Schlüssel. Gezählt und nummeriert werden die Verzeichnisse mittels der Summenvariablen „anz_dir“, die als Zähler fungiert. Als deklarierte Schlüsselreferenz wird jedoch eine andere Variable „no_dir“ verwendet, die beim Positionieren auf einzelne Einträge des Hash-Objektes

automatisch mit dem dort hinterlegten Datenwert überschrieben wird. Durch diese zusätzliche Schlüsselvariable wird das ungewollte Verstellen des Zählers „anz_dir“ verhindert. Die Definition eines Iterators ermöglicht die Navigation über das Hash-Objekt.

Initial wird nur das Startverzeichnis als Ausgangspunkt der Verarbeitung vermerkt.

Programmabschnitt 2: Verzeichnisanalyse als Verarbeitungsschleife

```
...
   /* 2. Verzeichnisanalyse: Arbeite Verzeichnisliste von hinten ab */
   stop = 0;
   DO WHILE (NOT stop);
      /* 2.1 Adressiere den letzten Verzeichniseintrag */
      ...
      /* 2.2 Ermittle und behandle die Member aus dem Verzeichnis */
      ...
      /* 2.2.1 Füge Verzeichnisse dem Hash-Objekt hinzu */
      ...
      /* 2.2.2 Gebe Dateien in SAS-Datei aus */
      ...
      /* 2.3 Lösche aktuellen Eintrag aus Hash-Objekt oder setze stop */
      ...
   END;
...
run;
```

Dargestellt ist zunächst nur der logische Ablauf. Innerhalb des Data Steps liegen vor der Schleifenverarbeitung folgende Informationen vor:

Vor Durchlauf 1	
Hash-Objekt	**SAS-Datei**
[1] D:\root	

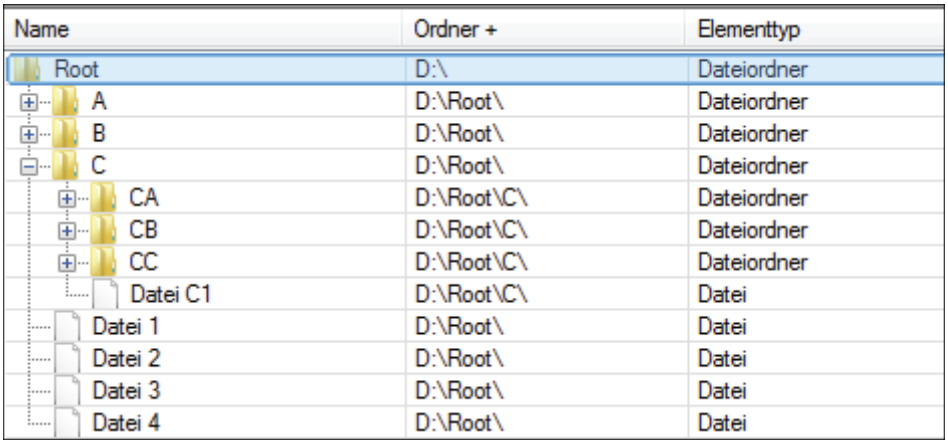
Im ersten Durchlauf der Schleife wird zunächst das Root-Verzeichnis mit den darin enthaltenen 3 Unterverzeichnissen und vier Dateien analysiert.

Name	Ordner +	Elementtyp
Root	D:\	Dateiordner
A	D:\Root\	Dateiordner
B	D:\Root\	Dateiordner
C	D:\Root\	Dateiordner
Datei 1	D:\Root\	Datei
Datei 2	D:\Root\	Datei
Datei 3	D:\Root\	Datei
Datei 4	D:\Root\	Datei

Im Ergebnis werden folgende Informationen abgelegt (die neu erkannten Unterverzeichnisse werden dem Hash-Objekt hinzugefügt, das vollständig ausgewertete Root-Verzeichnis wird als Eintrag gelöscht):

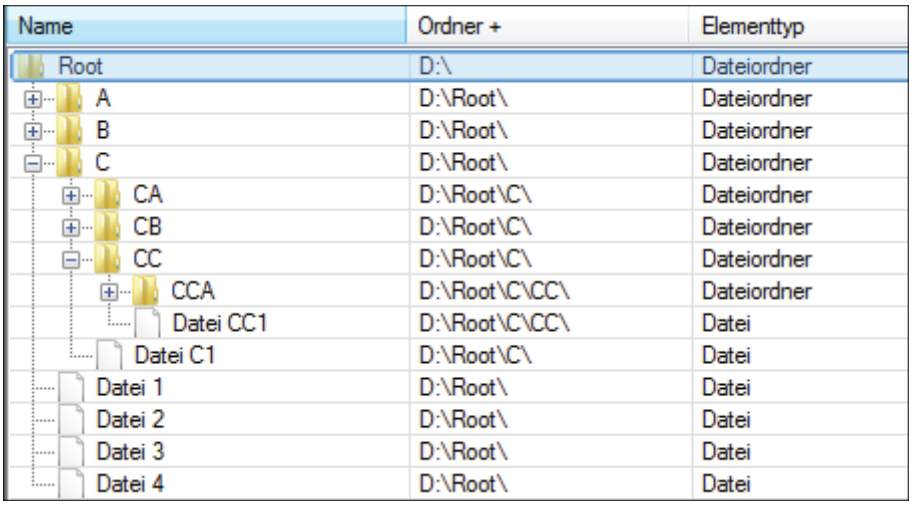
Nach Durchlauf 1	
Hash-Objekt	**SAS-Datei**
[2] d:\root\A	Datei 1
[3] d:\root\B	Datei 2
[4] d:\root\C	Datei 3
	Datei 4

In dem zweiten Durchlauf wird der letzte Eintrag des Hash-Objektes, also das Verzeichnis „d:\root\C“ untersucht.



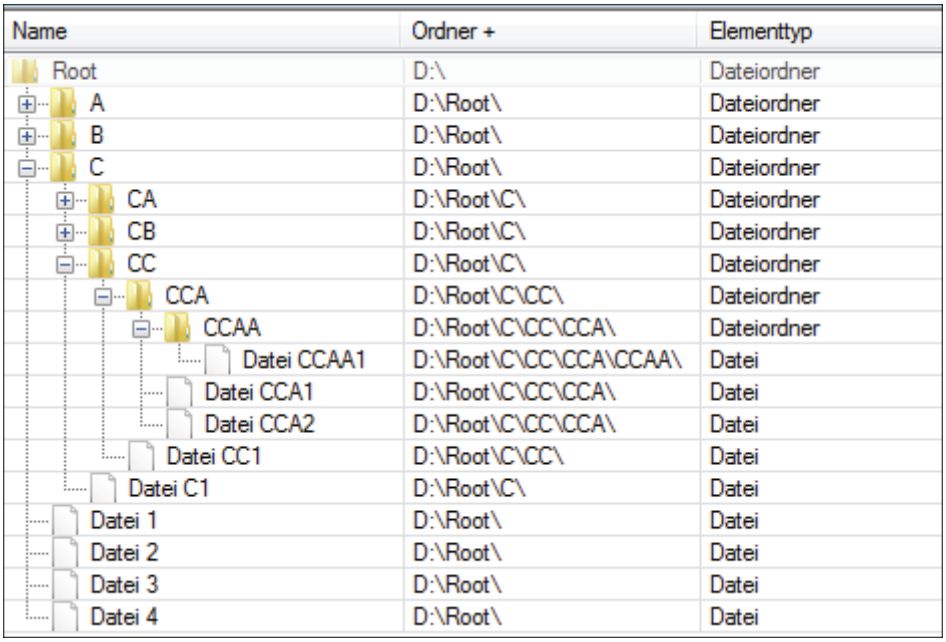
Nach Durchlauf 2	
Hash-Objekt	**SAS-Datei**
[2] d:\root\A	Datei 1
[3] d:\root\B	Datei 2
[5] d:\root\C\CA	Datei 3
[6] d:\root\C\CB	Datei 4
[7] d:\root\C\CC	Datei C1

Das nächste Objekt ist das Verzeichnis „d:\root\C\CC“.



Nach Durchlauf 3	
Hash-Objekt	**SAS-Datei**
[2] d:\root\A	Datei 1
[3] d:\root\B	Datei 2
[4] d:\root\C\CA	Datei 3
[5] d:\root\C\CB	Datei 4
[8] d:\root\C\CC\CCA	Datei C1
	Datei CC1

Nach diesem Prinzip werden schrittweise alle tiefer liegenden Unterverzeichnisse auf dort vorhandene Dateien überprüft, bis nach zwei weiteren Durchläufen „d:\root\C\CC\CCA\CCAA“ ohne weiteres Unterverzeichnis erreicht und ausgewertet ist. Da hiermit der Zweig „d:\root\C\CC“ mit all seinen Unterverzeichnissen vollständig abgearbeitet ist, beinhaltet das Hash-Objekt hierzu anschließend keine Einträge mehr. Es wird mit „d:\root\C\CB“ fortgesetzt.



Nach Durchlauf 5	
Hash-Objekt	**SAS-Datei**
[2] d:\root\A	Datei 1
[3] d:\root\B	Datei 2
[4] d:\root\C\CA	Datei 3
[5] d:\root\C\CB	Datei 4
[	Datei C1
	Datei CC1
	Datei CCA1
	Datei CCA2
	Datei CCAA1

Der dazu noch fehlende Programmcode ist nachfolgend dargestellt:

```
...
   /* 2. Verzeichnisanalyse: Arbeite Verzeichnisliste von hinten ab */
   stop = 0;
   DO WHILE (NOT stop);
      /* 2.1 Adressiere den letzten Verzeichniseintrag */
      h_dir_iter.LAST();
      remove_no_dir = no_dir;

      rc = FILENAME("indir", STRIP(path)!!"/");
      d_id = DOPEN("indir");

      /* 2.2 Ermittle und behandle die Member aus dem Verzeichnis */
      DO _j = 1 TO DNUM(d_id);
         member = DREAD(d_id, _j);
         member_path = STRIP(path)||"\"!!STRIP(member);

         /* Directory oder File ? */
         rc = FILENAME("infile", member_path);
         IF rc = 0 THEN DO;
            m_id = DOPEN("infile");
            /* 2.2.1 Füge Verzeichnisse dem Hash-Objekt hinzu */
            IF m_id > 0 THEN DO;    /* -> DIR */
               anz_dir + 1;
               h_dir.ADD(KEY: anz_dir, DATA: anz_dir, DATA: member_path);
               rc = DCLOSE(m_id);
            END;
            /* 2.2.1 Gebe Dateien in SAS-Datei aus */
            ELSE DO;
               m_id = FOPEN("infile");
               IF m_id > 0 THEN DO;       /* -> FILE */
                  anz_file + 1;
                  OUTPUT;
                  rc = FCLOSE(m_id);
               END;
            END;
         END;
      END;
      rc = DCLOSE(d_id);

      /* 2.3 Lösche Eintrag aus Hash-Objekt oder setze stop */
      IF h_dir.NUM_ITEMS = 1 THEN stop = 1;
      ELSE DO;
         h_dir_iter.FIRST();
         IF no_dir = remove_no_dir THEN h_dir_iter.LAST();
         h_dir.REMOVE(KEY: remove_no_dir);
      END;
   END;

   PUT "Summary: #Files: " anz_file COMMAX7. "    #Dirs: " anz_dir COMMAX7.;
RUN;
```

In das Log werden nachfolgende Summary-Informationen ausgegeben:

```
Summary: #Files:      25     #Dirs:       19
```

3.2.3 Erweiterungsvorschläge

Das Programmbeispiel ermittelt lediglich die Dateien aus dem Verzeichnisbaum und schreibt, die Dateinamen zusammen mit den Pfadangaben in eine SAS-Datei. Darauf aufbauend lassen sich folgende Erweiterungen auch innerhalb des obigen Data Steps realisieren:

- Vorgabe mehrerer Start-Verzeichnisse (z.B. Laufwerke) und von Exclude-Verzeichnissen.
- Einlesen weiterer Dateiattribute wie Size, Creation Date, Last Modified Date, …
- Analyse der ermittelten Dateien (auf potentielle Dubletten, Versionsstände, Backups, …)

4 Résumé

Die Beispielaufgaben sind auch auf anderem Wegen lösbar - erfordern dafür jedoch ggf. zusätzliche Verarbeitungsschritte (PROC und DATA)
Es genügen bereits sehr wenige ausgewählte Methoden um komplexe Aufgabenstellungen zu lösen.
Iteratoren erweitern die Einsatzmöglichkeiten signifikant.
Das Potential von Hash-Objekten ist derzeit noch weitgehend ungenutzt.

Literatur

[1] SAS 9.4 Language Reference: Concepts, Fourth Edition.

[2] Arne Leißner: „Geschwindigkeit ist nicht alles – wozu Hash Objekte wirklich gut sind“. In: A. Koch, R. Minkenberg (Hrsg.): KSFE 2015 - Proceedings der 19. Konferenz der SAS-Anwender in Forschung und Entwicklung (KSFE); Shaker Verlag, Aachen (2015), S. 189 - 207