# SepaTools

## DLL-Version

### API-Dokumentation

**84307 Eggenfelden, 30. Oktober 2026**

**G. Schliffenbacher**
**Tel.: 08721/911 926**
**Fax: 08721/10 456**
**Mail: mail@sepa-tools.de**

# *Inhaltsverzeichnis*

**1 Historie der Versionen ..... 14**

**2 Grundsätzliches ..... 26**
2.1 Datenstrukturen ..... 26
2.2 Returncodes ..... 26
2.3 DLL-Deklaration ..... 27
2.4 Debug-Version ..... 27
2.5 Datenbanken ..... 27
2.6 Datenbank_Mini ..... 28
2.7 Funktionen für Datenbank_Mini ..... 28
2.8 Datenbank_Big (Datenbank_Big.zip) ..... 28
2.9 Umlaute in SEPA XML-Dateien ..... 29

**3 Abweichungen und Hinweise für die 64-Bit Version ..... 30**
3.1 Dateinamen ..... 30
3.2 Datenstrukturen ..... 30
3.3 Einbindung der DLL ..... 30
3.4 Datenbanken ..... 30
3.5 SepaTools_Init ..... 30
3.6 Funktionsaufruf ..... 31
3.7 Parameter ..... 31
3.8 Returncodes ..... 31
3.9 Versionsführung ..... 32
3.10 Performance-Tests ..... 32

**4 Bankfachliche Grundlagen ..... 33**

**5 Testprogramm DLLDemo.exe ..... 35**

**6 SepaTools_Version ..... 36**
6.1 Zweck der Funktion ..... 36
6.2 Funktionsaufruf ..... 36
6.3 Parameter ..... 36
6.4 Returncodes ..... 36

**7 SepaTools_SetLogFile ..... 37**
7.1 Zweck der Funktion ..... 37
7.2 Funktionsaufruf ..... 37
7.3 Parameter ..... 37
7.4 Returncodes ..... 39
7.5 Beispiel einer kleinen Log-Datei ..... 39

**8 SepaTools_SetLogFileInhalt ..... 40**
8.1 Zweck der Funktion ..... 40
8.2 Funktionsaufruf ..... 40
8.3 Parameter ..... 40
8.4 Returncodes ..... 40

**9 SepaTools_Init ..... 41**

9.1 Zweck der Funktion ..... 41
9.2 Funktionsaufruf ..... 41
9.3 Parameter ..... 41
9.4 Returncodes ..... 42

**10 SepaTools_Free ..... 44**
10.1 Zweck der Funktion ..... 44
10.2 Funktionsaufruf ..... 44
10.3 Parameter ..... 44
10.4 Returncodes ..... 44

**11 SepaTools_SetOptions ..... 45**
11.1 Zweck der Funktion ..... 45
11.2 Funktionsaufruf ..... 45
11.3 Parameter ..... 45
11.4 Returncodes ..... 51

**12 SepaTools_Info ..... 52**
12.1 Zweck der Funktion ..... 52
12.2 Funktionsaufruf ..... 52
12.3 Parameter ..... 52
12.4 Returncodes ..... 52

**13 XML-Dateien erzeugen ..... 53**
13.1 Automatische Bildung von Sammlern ..... 53

**14 SepaTools_CreateXML ..... 54**
14.1 Zweck der Funktion ..... 54
14.2 Funktionsaufruf ..... 54
14.3 Parameter ..... 54
14.4 Returncodes ..... 55

**15 SepaTools_CreateXMLExt ..... 56**
15.1 Zweck der Funktion ..... 56
15.2 Funktionsaufruf ..... 56
15.3 Parameter ..... 56
15.4 Returncodes ..... 58

**16 SepaTools_SetContactDetails ..... 59**
16.1 Zweck der Funktion ..... 59
16.2 Parameter ..... 59
16.3 Returncodes ..... 61

**17 SepaTools_WriteXML ..... 62**
17.1 Zweck der Funktion ..... 62
17.2 Funktionsaufruf ..... 62
17.3 Parameter ..... 62
17.4 Returncodes ..... 64

**18 SepaTools_WriteXMLExt ..... 68**
18.1 Zweck der Funktion ..... 68
18.2 Funktionsaufruf ..... 68
18.3 Parameter ..... 68

18.4 Returncodes (zusätzlich) ..... 74

**19 SepaTools_CloseXML ..... 75**
19.1 Zweck der Funktion ..... 75
19.2 Funktionsaufruf ..... 75
19.3 Parameter ..... 75
19.4 Returncodes ..... 75

**20 XML-Unterstützung für die Schweiz ..... 77**
20.1 Überweisungen ..... 77
20.1.1 Inlandszahlungen ..... 78
20.1.2 Auslandszahlungen ..... 79
20.2 Lastschriften ..... 80
20.3 Plausibilitätsprüfungen ..... 81
20.4 Validierung der XML-Dateien ..... 81
20.5 Implementierung ..... 81

**21 SepaTools_XMLSetId ..... 82**
21.1 Zweck der Funktion ..... 82
21.2 Funktionsaufruf ..... 82
21.3 Parameter ..... 82

**22 SepaTools_XMLGetData ..... 84**
22.1 Zweck der Funktion ..... 84
22.2 Funktionsaufruf ..... 84
22.3 Parameter ..... 84
22.4 Returncodes ..... 85

**23 SepaTools_XMLFreeData ..... 86**
23.1 Zweck der Funktion ..... 86
23.2 Funktionsaufruf ..... 86
23.3 Parameter ..... 86
23.4 Returncodes ..... 86

**24 DTA-Dateien in XML-Dateien umsetzen ..... 87**

**25 SepaTools_ReadDTA ..... 89**
25.1 Zweck der Funktion ..... 89
25.2 Funktionsaufruf ..... 89
25.3 Parameter ..... 89
25.4 Returncodes ..... 90

**26 SepaTools_GetDTAFehler ..... 91**
26.1 Zweck der Funktion ..... 91
26.2 Funktionsaufruf ..... 91
26.3 Parameter ..... 91
26.4 Returncodes ..... 92

**27 SepaTools_GetDTAInfo ..... 93**
27.1 Zweck der Funktion ..... 93
27.2 Funktionsaufruf ..... 93
27.3 Parameter ..... 93
27.4 Returncodes ..... 94

**28 SepaTools_PutDTAInfo ..... 95**
28.1 Zweck der Funktion ..... 95
28.2 Funktionsaufruf ..... 95
28.3 Parameter ..... 95
28.4 Returncodes ..... 99

**29 SepaTools_WriteDTAtoXML ..... 100**
29.1 Zweck der Funktion ..... 100
29.2 Funktionsaufruf ..... 100
29.3 Parameter ..... 100
29.4 Returncodes ..... 101

**30 SepaTools_GetDTAProtokoll ..... 102**
30.1 Zweck der Funktion ..... 102
30.2 Funktionsaufruf ..... 102
30.3 Parameter ..... 102
30.4 Detail-Fehlercodes ..... 103
30.5 Returncodes ..... 104

**31 SepaTools_FreeDTAVars ..... 105**
31.1 Zweck der Funktion ..... 105
31.2 Parameter ..... 105
31.3 Returncodes ..... 105
31.4 Ablaufdiagramm ..... 106
31.5 Beschreibung des Ablaufdiagramms ..... 108

**32 DTAZV-Dateien in XML-Dateien umsetzen ..... 110**

**33 SepaTools_ConvertDTAZVtoXML ..... 111**
33.1 Zweck der Funktion ..... 111
33.2 Funktionsaufruf ..... 111
33.3 Parameter ..... 111
33.4 Returncodes ..... 112

**34 SepaTools_GetDTAProtokoll ..... 113**
34.1 Zweck der Funktion ..... 113
34.2 Funktionsaufruf ..... 113
34.3 Parameter ..... 113
34.4 Detail-Fehlercodes ..... 113
34.5 Returncodes ..... 113

**35 CSV-Dateien in XML-Dateien umsetzen ..... 114**

**36 SepaTools_ConvertCSVtoXML ..... 115**
36.1 Zweck der Funktion ..... 115
36.2 Funktionsaufruf ..... 115
36.3 Parameter ..... 115
36.4 Returncodes ..... 121

**37 SepaTools_GetDTAProtokoll ..... 122**
37.1 Zweck der Funktion ..... 122
37.2 Funktionsaufruf ..... 122
37.3 Parameter ..... 122

37.4 Detail-Fehlercodes ..... 122
37.5 Returncodes ..... 122
37.6 Beispiel für die CSV-Konvertierung ..... 122
37.6.1 Belegung der Struktur ..... 123

**38 DTAZV-Dateien erzeugen ..... 124**

**39 SepaTools_CreateAZV ..... 125**
39.1 Zweck der Funktion ..... 125
39.2 Funktionsaufruf ..... 125
39.3 Parameter ..... 125
39.4 Returncodes ..... 126

**40 SepaTools_WriteAZV ..... 128**
40.1 Zweck der Funktion ..... 128
40.2 Funktionsaufruf ..... 128
40.3 Parameter ..... 128
40.4 Returncodes ..... 129

**41 SepaTools_CloseAZV ..... 132**
41.1 Zweck der Funktion ..... 132
41.2 Funktionsaufruf ..... 132
41.3 Parameter ..... 132
41.4 Returncodes ..... 132

**42 SepaTools_GetLandWhg ..... 133**
42.1 Zweck der Funktion ..... 133
42.2 Funktionsaufruf ..... 133
42.3 Parameter ..... 133
42.4 Returncodes ..... 134

**43 SepaTools_SetVersionUndLand ..... 135**
43.1 Zweck der Funktion ..... 135
43.2 Funktionsaufruf ..... 135
43.3 Parameter ..... 135
43.4 Returncodes ..... 137

**44 SepaTools_XMLLesenInit ..... 138**
44.1 Zweck der Funktion ..... 138
44.2 Funktionsaufruf ..... 138
44.3 Parameter ..... 138
44.4 Returncodes ..... 138

**45 SepaTools_XMLLesen ..... 140**
45.1 Zweck der Funktion ..... 140
45.2 Funktionsaufruf ..... 140
45.3 Parameter ..... 140
45.4 Returncodes ..... 141
45.5 Hinweise ..... 141

**46 SepaTools_XMLLesenClose ..... 143**
46.1 Zweck der Funktion ..... 143
46.2 Funktionsaufruf ..... 143

46.3 Parameter ..... 143
46.4 Returncodes ..... 143

**47 SepaTools_GetBIC ..... 144**
47.1 Zweck der Funktion ..... 144
47.2 Funktionsaufruf ..... 144
47.3 Parameter ..... 144
47.4 Returncodes ..... 144

**48 SepaTools_CheckIBAN ..... 145**
48.1 Zweck der Funktion ..... 145
48.2 Funktionsaufruf ..... 145
48.3 Parameter ..... 145
48.4 Returncodes ..... 145

**49 SepaTools_CheckIBAN_Strong ..... 146**
49.1 Zweck der Funktion ..... 146
49.2 Funktionsaufruf ..... 146
49.3 Parameter ..... 146
49.4 Returncodes ..... 146

**50 SepaTools_GetBLZKonto ..... 147**
50.1 Zweck der Funktion ..... 147
50.2 Funktionsaufruf ..... 147
50.3 Parameter ..... 147
50.4 Returncodes ..... 148

**51 SepaTools_CheckIBANLand ..... 150**
51.1 Zweck der Funktion ..... 150
51.2 Funktionsaufruf ..... 150
51.3 Parameter ..... 150
51.4 Returncodes ..... 150

**52 SepaTools_CheckCI ..... 151**
52.1 Zweck der Funktion ..... 151
52.2 Funktionsaufruf ..... 151
52.3 Parameter ..... 151
52.4 Returncodes ..... 151
52.5 Bekannte Längen von CIs ..... 151

**53 SepaTools_CheckESR ..... 153**
53.1 Zweck der Funktion ..... 153
53.2 Funktionsaufruf ..... 153
53.3 Parameter ..... 153
53.4 Returncodes ..... 153

**54 SepaTools_GetESRPrZiff ..... 154**
54.1 Zweck der Funktion ..... 154
54.2 Funktionsaufruf ..... 154
54.3 Parameter ..... 154
54.4 Returncodes ..... 154

**55 SepaTools_CheckSEPATeilnahme ..... 155**

55.1 Zweck der Funktion ..... 155
55.2 Funktionsaufruf ..... 155
55.3 Parameter ..... 155
55.4 Returncodes ..... 155

**56 SepaTools_ConvertBLZKonto ..... 156**
56.1 Zweck der Funktion ..... 156
56.2 Funktionsaufruf ..... 156
56.3 Parameter ..... 156
56.4 Returncodes ..... 157

**57 SepaTools_SetErfahrung ..... 159**
57.1 Zweck der Funktion ..... 159
57.2 Funktionsaufruf ..... 159
57.3 Parameter ..... 159
57.4 Returncodes ..... 159

**58 SepaTools_GetBankInfo ..... 160**
58.1 Zweck der Funktion ..... 160
58.2 Funktionsaufruf ..... 160
58.3 Parameter ..... 160
58.4 Returncodes ..... 161

**59 SepaTools_GetBICInfo ..... 162**
59.1 Zweck der Funktion ..... 162
59.2 Funktionsaufruf ..... 162
59.3 Parameter ..... 162
59.4 Returncodes ..... 163

**60 SepaTools_GetIBANLand ..... 164**
60.1 Zweck der Funktion ..... 164
60.2 Funktionsaufruf ..... 164
60.3 Parameter ..... 164
60.4 Returncodes ..... 164

**61 SepaTools_GetSEPABank ..... 165**
61.1 Zweck der Funktion ..... 165
61.2 Funktionsaufruf ..... 165
61.3 Parameter ..... 165
61.4 Returncodes ..... 166

**62 SepaTools_GetBLZ ..... 167**
62.1 Zweck der Funktion ..... 167
62.2 Funktionsaufruf ..... 167
62.3 Parameter ..... 167
62.4 Returncodes ..... 168

**63 SepaTools_FindBLZ ..... 169**
63.1 Zweck der Funktion ..... 169
63.2 Funktionsaufruf ..... 169
63.3 Parameter ..... 169
63.4 Returncodes ..... 170

**64 SepaTools_SearchBLZ ..... 171**
64.1 Zweck der Funktion ..... 171
64.2 Funktionsaufruf ..... 171
64.3 Parameter ..... 171
64.4 Returncodes ..... 172

**65 SepaTools_CheckPruefziffer ..... 173**
65.1 Zweck der Funktion ..... 173
65.2 Funktionsaufruf ..... 173
65.3 Parameter ..... 173
65.4 Returncodes ..... 173

**66 SepaTools_GetPurposeCode ..... 174**
66.1 Zweck der Funktion ..... 174
66.2 Funktionsaufruf ..... 174
66.3 Parameter ..... 174
66.4 Returncodes ..... 175

**67 SepaTools_SearchPurposeCodes ..... 176**
67.1 Zweck der Funktion ..... 176
67.2 Funktionsaufruf ..... 176
67.3 Returncodes ..... 177

**68 SepaTools_GetReasonCode ..... 178**
68.1 Zweck der Funktion ..... 178
68.2 Funktionsaufruf ..... 178
68.3 Parameter ..... 178
68.4 Returncodes ..... 178

**69 SepaTools_SearchReasonCodes ..... 179**
69.1 Zweck der Funktion ..... 179
69.2 Funktionsaufruf ..... 179
69.3 Returncodes ..... 180

**70 SepaTools_AddUserBIC ..... 181**
70.1 Zweck der Funktion ..... 181
70.2 Funktionsaufruf ..... 181
70.3 Parameter ..... 181
70.4 Returncodes ..... 181

**71 SepaTools_DelUserBIC ..... 182**
71.1 Zweck der Funktion ..... 182
71.2 Funktionsaufruf ..... 182
71.3 Parameter ..... 182
71.4 Returncodes ..... 182

**72 SepaTools_GetUserBIC ..... 183**
72.1 Zweck der Funktion ..... 183
72.2 Funktionsaufruf ..... 183
72.3 Parameter ..... 183
72.4 Returncodes ..... 183

**73 SepaTools_FindUserBIC ..... 184**

73.1 Zweck der Funktion ..... 184
73.2 Funktionsaufruf ..... 184
73.3 Parameter ..... 184
73.4 Returncodes ..... 184

**74 SepaTools_AddUserIBAN ..... 185**
74.1 Zweck der Funktion ..... 185
74.2 Funktionsaufruf ..... 185
74.3 Parameter ..... 185
74.4 Returncodes ..... 185

**75 SepaTools_DelUserIBAN ..... 186**
75.1 Zweck der Funktion ..... 186
75.2 Funktionsaufruf ..... 186
75.3 Parameter ..... 186
75.4 Returncodes ..... 186

**76 SepaTools_GetUserIBAN ..... 188**
76.1 Zweck der Funktion ..... 188
76.2 Funktionsaufruf ..... 188
76.3 Parameter ..... 188
76.4 Returncodes ..... 188

**77 SepaTools_FindUserIBAN ..... 189**
77.1 Zweck der Funktion ..... 189
77.2 Funktionsaufruf ..... 189
77.3 Parameter ..... 189
77.4 Returncodes ..... 189

**78 SepaTools_GetTargetDatum ..... 190**
78.1 Zweck der Funktion ..... 190
78.2 Funktionsaufruf ..... 190
78.3 Parameter ..... 190
78.4 Returncodes ..... 191

**79 SepaTools_IsTargetDatum ..... 192**
79.1 Zweck der Funktion ..... 192
79.2 Funktionsaufruf ..... 192
79.3 Parameter ..... 192
79.4 Returncodes ..... 192

**80 Pain.007-Dateien erzeugen ..... 193**

**81 SepaTools_CreateXML007 ..... 194**
81.1 Zweck der Funktion ..... 194
81.2 Funktionsaufruf ..... 194
81.3 Parameter ..... 194
81.4 Returncodes ..... 195

**82 SepaTools_WriteXML007 ..... 196**
82.1 Zweck der Funktion ..... 196
82.2 Funktionsaufruf ..... 196
82.3 Parameter ..... 196

82.4 Returncodes ..... 197

**83 SepaTools_CloseXML007 ..... 199**
83.1 Zweck der Funktion ..... 199
83.2 Funktionsaufruf ..... 199
83.3 Parameter ..... 199
83.4 Returncodes ..... 199

**84 CGI-Dateien erzeugen ..... 200**

**85 SepaTools_CreateCGI ..... 201**
85.1 Zweck der Funktion ..... 201
85.2 Funktionsaufruf ..... 201
85.3 Parameter ..... 201
85.4 Returncodes ..... 202

**86 SepaTools_WriteCGI ..... 204**
86.1 Zweck der Funktion ..... 204
86.2 Funktionsaufruf ..... 204
86.3 Parameter ..... 204
86.4 Returncodes ..... 207

**87 SepaTools_CloseCGI ..... 209**
87.1 Zweck der Funktion ..... 209
87.2 Funktionsaufruf ..... 209
87.3 Parameter ..... 209
87.4 Returncodes ..... 209

**88 Neues AZV-Format (XML) ab 2022, zwingend ab 2025 ..... 211**
88.1 Abgrenzung zum CGI-Format ..... 211
88.2 Bankfachliche Aspekte ..... 211
88.2.1 Service-Level ..... 211
88.2.2 Codes zum Tragen von Entgelten ..... 212
88.2.3 Zulässige Scheck Typen ..... 212
88.2.4 Zulässige Scheck-Lieferarten ..... 212
88.2.5 Weisungsschlüssel ..... 213
88.2.6 Codes für den strukturierten Verwendungszweck ..... 213

**89 AZV Neu Dateien erzeugen ..... 214**
89.1 Abgrenzung zum CGI-Format ..... 214

**90 Verwendung von Adressen ..... 215**

**91 SepaTools_CreateAZVN ..... 216**
91.1 Zweck der Funktion ..... 216
91.2 Funktionsaufruf ..... 216
91.3 Parameter ..... 216
91.4 Returncodes ..... 218

**92 SepaTools_WriteAZVN ..... 219**
92.1 Zweck der Funktion ..... 219
92.2 Funktionsaufruf ..... 219
92.3 Parameter ..... 219

92.4 Returncodes ..... 225

**93 SepaTools_CloseAZVN ..... 227**
93.1 Zweck der Funktion ..... 227
93.2 Funktionsaufruf ..... 227
93.3 Parameter ..... 227
93.4 Returncodes ..... 227

**94 SepaTools_XMLGetDataAZVN ..... 229**
94.1 Zweck der Funktion ..... 229
94.2 Funktionsaufruf ..... 229
94.3 Parameter ..... 229
94.4 Returncodes ..... 230

**95 Auslesen von Camt- und Pain.002-Nachrichten ..... 231**

**96 SepaTools_ConvertCamtToCSV ..... 232**
96.1 Zweck der Funktion ..... 232
96.2 Funktionsaufruf ..... 232
96.3 Parameter ..... 232
96.4 Vorgabe zur Belegung des Feldes <BkTxCd><Prtry><Cd> ..... 246
96.5 Returncodes ..... 247
96.6 Verarbeitung von ZIP-Dateien ..... 247

**97 Camt-Dateien und Pain.002-Dateien auslesen ..... 249**

**98 SepaTools_CamtOpen ..... 250**
98.1 Zweck der Funktion ..... 250
98.2 Funktionsaufruf ..... 250
98.3 Parameter ..... 250
98.4 Returncodes ..... 252

**99 SepaTools_CamtRead ..... 253**
99.1 Zweck der Funktion ..... 253
99.2 Funktionsaufruf ..... 253
99.3 Parameter ..... 253
99.4 Returncodes ..... 259

**100 SepaTools_CamtClose ..... 260**
100.1 Zweck der Funktion ..... 260
100.2 Funktionsaufruf ..... 260
100.3 Parameter ..... 260
100.4 Returncodes ..... 260

**101 Verarbeitung von ZIP-Dateien ..... 261**

**102 Camt.053 und Camt.054 Dateien schreiben ..... 262**
102.1 SepaTools_CamtWriteOpen ..... 262
102.2 Zweck der Funktion ..... 262
102.3 Funktionsaufruf ..... 262
102.4 Parameter ..... 262
102.5 Returncodes ..... 264

**103 SepaTools_CamtWrite ..... 265**
103.1 Zweck der Funktion ..... 265
103.2 Funktionsaufruf ..... 265
103.3 Parameter ..... 265
103.4 Returncodes ..... 266

**104 SepaTools_CamtWriteClose ..... 267**
104.1 Zweck der Funktion ..... 267
104.2 Funktionsaufruf ..... 267
104.3 Parameter ..... 267
104.4 Returncodes ..... 267
104.5 Praktische Hinweise ..... 267

**105 DTA/DTI-Dateien in Camt.054-Dateien umsetzen ..... 268**
105.1 Zweck der Funktion ..... 268
105.2 Funktionsaufruf ..... 268
105.3 Parameter ..... 268
105.4 Returncodes ..... 268

**106 SepaTools_SetDTIMapping ..... 270**
106.1 Zweck der Funktion ..... 270
106.2 Funktionsaufruf ..... 270
106.3 Parameter ..... 270
106.4 Returncodes ..... 271

**107 SepaTools_ConvertMT940 ..... 272**
107.1 Zweck der Funktion ..... 272
107.2 Funktionsaufruf ..... 272
107.3 Parameter ..... 272
107.4 Returncodes ..... 273

**108 SepaTools_ConvertCamt54ToDTI ..... 274**
108.1 Zweck der Funktion ..... 274
108.2 Funktionsaufruf ..... 274
108.3 Parameter ..... 274
108.4 Returncodes ..... 278

**109 SepaTools_ConvertCamt53ToMT940 ..... 280**
109.1 Zweck der Funktion ..... 280
109.2 Funktionsaufruf ..... 280
109.3 Parameter ..... 280
109.4 Returncodes ..... 282

**110 SepaTools_ConvertCamt53ToMT940Generic ..... 284**
110.1 Zweck der Funktion ..... 284
110.2 Funktionsaufruf ..... 284
110.3 Parameter ..... 284
110.4 Returncodes ..... 286

# 1 Historie der Versionen

Hier werden die wesentlichen Änderungen der Dokumentation chronologisch fortgeschrieben.

| Version | Datum | Dokumentänderungen |
|---|---|---|
| 3.90.0 | 01.11.2026 | - Ab November 2026 werden grundsätzlich strukturierte Adressen gefordert. Dies wird von SepaTools schon seit mehreren Jahren unterstützt. Unstrukturierte Adressangaben sind nicht mehr möglich.<br><br>- Unterstützung der Pain-Versionen pain.001.001.09 und pain.008.001.08.<br><br>- Bei der Konvertierung von Camt-Dateien in MT940 wird bei Währungsbuchungen wie bisher der in Euro konvertierte Betrag und zusätzlich im Verwendungszweck informativ der Ursprungsbetrag mit der Währungsangabe übergeben.<br><br>Die Darstellung des Tags </SvcLvl> innerhalb der XML-Datei war für Instantzahlungen nicht korrekt. Dieses Problem wurde behoben.<br><br><SvcLvl><br><Cd>SEPA</Cd><br></SvcLvl><br><LclInstrm><br><Cd>INST</Cd><br></LclInstrm><br><br>- Bei der Konvertierung von DTAZV-Zahlungen nach XML wurde ein Problem bzgl. der PmtTpInfo bereinigt. Die PmtTpInfo kann jetzt sowohl im PmtInf-Bereich als auch im CdtTrfTxInf-Bereich stehen.<br><br>Beide Varianten sind gültig. Es gibt aber Banken, die diese Position vorgeben.<br><br>- Bei der Darstellung des Verwendungszwecks bei der Konvertierung von Camt-Kontoauszügen wurden Optimierungen vorgenommen.<br><br>- Bei Konvertierungen von Fremdwährungszahlungen von Camt-Kontoauszügen nach MT940 wird der Fremdwährungsbetrag in den Verwendungszweck übernommen.<br><br>Für den Betrag wurde fälschlicherweise der Währungsbetrag und nicht der Euro-Betrag aus den „Refs“ übernommen.<br><br>- Bei den Prüfziffernberechnungen gab es KEINE Änderungen.<br><br>- Werden IBAN Only Zahllungen erzeugt, so wurde bisher ein leeres Tag für das Kreditinstitut des Zahlungsempfängers ausgegeben. Jetzt wird das Tag erst gar nicht ausgegeben.<br><br>- Das Tag BICFI wird jetzt durchgängig anstelle des Tags BIC verwendet.<br><br>- Der Timestamp bei Zahlungsverkehrsdateien wird jetzt entspre- |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | chend der internationalen Regelungen ausgegeben. Der Wert für die Millisekunden wird nun mit .001Z angegeben.<br><br>- Für den Schweizer Zahlungsverkehr wird wieder eine Bankclearingnummer übergeben, soweit diese vorliegt.<br><br>- Bisher wurde bei einem fehlenden Ort das Feld „TwnNm“ mit dem Wert „Unbekannt“ gefüllt. Jetzt wird dieses Tag, bei fehlendem Wert, gar nicht geschrieben. |
| 3.88.0 | 28.05.2026 | In der XMLOptionStruct wurde das zusätzliche Feld ***PmtTpInfo*** eingefügt.<br><br>Wenn Sie dieses Feld mit dem Wert „1“ belegen, so wird die Payment Type Info bei Auslandszahlungen anstatt in den Payment Info Bereich in den Transaktionsbereich geschrieben wird. |
| 3.85.5 | 16.11.2025 | Die neue Funktion SepaTools_ConvertCamt53ToMT940Generic ermöglicht es die Parameter einzeln zu übergeben, anstelle einer Struktur.<br><br>Beim Buchungsdatum in einer Camt-Datei wird ein alternatives Format ausgelesen. Für den Anwender und/oder Programmierer besteht kein Handlungsbedarf. |
| 3.85.6 | 03.08.2025 | Die Funktion SepaTools_SetNewUltimate ist entfallen. Es gibt dafür innerhalb der Optionen (DLL-Version) den neuen Wert „NewUltimate“ und in der Batch-Version den neuen Schlüsselbegriff „Ultimate“ (ggfls. Belegung mit dem Wert „1“).<br><br>Damit wird der abweichende Auftraggeber im Transaktionsblock und nicht in der Payment Information dargestellt.<br><br>Bei der Konvertierung von einer CSV-Datei in eine DTAZV-Datei (Basis-Version) wurde ein Fehler bereinigt.<br><br>Die Schnittstellenbeschreibung für die Konvertierung von CSV-Dateien zu DTAZV-Dateien wurde um ein Feld (Empf_Land) ergänzt.<br><br>In der Version ist der aktuelle Datenbestand enthalten. |
| 3.85.3 | 18.07.2025 | Es erfolgten kleinere Fehlerbereinigungen. Der aktuelle Bankleitzahlenbestand der Deutschen Bundesbank steht zur Verfügung. |
| 3.85.2 | 11.05.2025 | Es gibt innerhalb der Optionen das Feld „NewUltimate“. Dieses dient der Darstellung des Abweichenden Auftraggebers der Zahlung. Dieser wird normalerweise im Payment Information Block dargestellt.<br><br>Wenn Sie hier den Wert mit „1“ belegen, so wird der abweichende Auftraggeber im Transaktionsblock angegeben. Dies ist hilfreich, wenn Sie für jede Zahlung einen unterschiedlichen Auftraggeber haben. |
| 3.82 | 03.08.2023 | Bitte benutzen Sie bei allen Adressangaben die „Struct“ Variablen. |
| 3.80 | 06.05.2023 | - In der Dokumentation der Struktur CamtReadStruct wurde ein Fehler behoben. Das Feld PainAddtInfo (sechst letztes Feld in der Struktur) hat in der Doku gefehlt und wurde nun ergänzt. |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | - Bei Überweisungen wird die Version ***pain.001.001.09*** unterstützt. Damit ist es auch möglich für Instant Überweisungen neben dem Ausführungsdatum auch eine Ausführungszeit mit zu geben.<br><br>- Bei der Konvertierung von DTA-Dateien in XML-Dateien, wird die Ausführungszeit im Rahmen der Funktion ***SepaTools_PutDTAInfo*** übergeben. Dazu wurde die Struktur ***DTAInfoStruct*** innerhalb des Reservebereichs um ein Feld erweitert.<br><br>- Um bei der Konvertierung von CSV-Dateien auch die Uhrzeit der Ausführung übergeben zu können, wurde die Struktur ***CSVConvertStruct*** im Reservebereich erweitert. Die Übergabe innerhalb der CSV-Datei ist aber auch möglich. Siehe hierzu die Doku.<br><br>- Bei Auslandsüberweisungen im XML-Format sind nun jetzt auch Scheckzahlungen möglich.<br><br>- In Einzelfällen ist es vorgekommen, dass Banken bei der Valuta das ungültige Datum 30. Februar ausgeben. Mit diesem ungültigen Datum kann SepaTools nicht umgehen. In diesen Fällen wird das Datum von der Funktion ***SepaTools_ConvertMT940*** intern auf den 28. oder 29. Februar (je nach Schaltjahr) gesetzt.<br><br>- Ansonsten sollten nur die Datenbanken ausgetauscht werden. |
| 3.70 | 09.02.2023 | Die Struktur ***CamtReadStruct*** wurde am Ende der Struktur im Reservebereich um das Feld ***DateiName*** ergänzt. Dieses Feld enthält dann den Namen der gerade ausgewerteten XML-Datei. Zu den Details siehe bitte die Dokumentation.<br><br>Die Prüfung von MT940-Dateien wurde erweitert (etwas lockerer). |
| 3.61 | 04.11.2022 | In die Datenstruktur ***CamtReadStruct*** sind ganz hinten im Reservebereich drei Felder für zusätzliche Informationen beim strukturierten Verwendungszweck eingefügt worden.<br><br>In der Datenstruktur ***XMLWriteExtStruct*** wurde das Feld ***CHQRInfo*** (ganz zum Schluss) im Reservebereich ergänzt. Dort können ergänzende Informationen für den strukturierten Verwendungszweck bei Schweizer QR-Zahlungen untergebracht werden. |
| 3.60 | 06.08.2022 | Der Auslandszahlungsverkehr nach ISO 20022 wurde an die ab November geltende DK-Version 3.6 angepasst. |
| 3.50 | 06.05.2022 | - Es gibt Funktionen zum Schreiben von Camt.053 und Camt.054 Dateien (analog wie das Lesen dieser Dateien).<br><br>Siehe hierzu die Doku.<br><br>- Es gibt die neuen Funktionen SepaTools_CheckESR und SepaTools_GetESRPrZiff. Damit können die Prüfziffer für Schweizer ESR-Nummern geprüft, bzw. berechnet werden.<br><br>Siehe hierzu die Dokumentation. |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | - Besondere Hinweise für die 64-Bit Version:<br><br>In der 64-Bit Version wurde ein Aufruffehler bei der Funktion SepaTools_GetPurposeCode bereinigt.<br><br>Die Darstellung des <AmdmntInd> Tags ist jetzt in der 64-Bit Version identisch mit der 32-Bit Version.<br><br>- Bei der Umsetzung von Camt-Dateien in eine MT940-Datei wird jetzt auch der strukturierte Verwendungszweck ausgewertet. |
| 3.40 | 09.02.2022 | - Beim deutschen Prüfziffernverfahren wurde im Verfahren A4 ein Fehler bereinigt.<br><br>- Bei der Funktion SepaTools_CheckIBAN wurden die Rückgabewerte in Teilen geändert. Bitte prüfen Sie in der Dokumentation, ob das auf Sie Auswirkungen hat.<br><br>- Bei der Funktion SepaTools_GetBLZKonto kamen neue Rückgabewertet hinzu. Siehe hierzu die Doku. |
| 3.30 | 06.11.2021 | - Innerhalb der Funktion ***SepaTools_CamtRead*** gibt das neue Feld ***SammlerSumme*** im Reservebereich. Darin wird bei der Einzelauflistung der Buchungen eines Sammlers, die Summe des Sammlers ausgegeben.<br><br>- Mit der Funktion ***SepaTools_ConvertCamtToCSV*** kann bei Ausgabe der Einzelbuchungen auch die Summe des Sammlers ausgeben werden. Siehe hierzu die Doku. |
| 3.20 | 03.08.2021 | - Über die neuen Funktiionen ***SepaTools_GetReasonCode*** und ***SepaTools_SearchReasonCodes*** können Reason Codes abgefragt werden.<br><br>- Es gibt die optionale Funktion ***SepaTools_SetContactDetails***. Siehe hierzu die Doku.<br><br>- In der Funktion ***SepaTools_ConvertCSVToXML*** kann jetzt auch ein Wert für ggfls. zu überlesende Kopfzeilen angegeben werden.<br><br>Ebenso unterstützt die Funktion nun auch IBAN Only, wenn eine IBAN übergeben wird.<br><br>- Die Funktionen ***SepaTools_CheckIBAN*** und ***SepaTools_CheckIBAN_Strong*** liefern zwei zusätzliche Rückgabecodes. Siehe hierzu bitte die Doku.<br><br>- Die Rückgabecodes der Funktion ***SepaTools_GetBLZKonto*** wurden um zwei bitcodierte positive Werte erweitert. Siehe hierzu bitte die Doku. |
| 3.10 | 05.05.2021 | - Die Struktur XMLOptionStruct wurde im Reservebereich um die die Felder MT940KontoMinLen und MT940MoreBuTag erweitert. Siehe hierzu die Doku.<br><br>- Derzeit wird in Deutschland die ISO Version von 2009 für das Camt- |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | Format verwendet. Ab dem November 2021 wird die Deutsche Kreditwirtschaft auf das Format 2019 nach ISO 20022 umsteigen.<br><br>In der Struktur XMLOptionStruct gibt es das zusätzliche Feld Camt-Version. Siehe hierzu bitte die Doku.<br><br>Zur Information:<br><br>Wenn Sie nur Camt-Dateien auslesen wollen, so haben Sie keinen Handlungsbedarf. Die unterschiedlichen Formate werden automatisch erkannt.<br><br>- Die Struktur XMLWriteExtStruct wurde im Reservebereich erweitert (Felder EmpfAdrHausnummer und folgende). Siehe hierzu bitte die Doku.<br><br>Wenn Sie diese neuen Felder nicht verwenden wollen, so besteht kein Handlungsbedarf.<br><br>- Wichtiger Hinweis für Anwender, die Zahlungen in der Schweiz einreichen:<br><br>Die Struktur für die Adresse der Bank des Zahlungsempfängers in der Schweiz wurde geändert. Die Änderungen betreffen Sie nur, wenn Sie für Einreichungen in der Schweiz die Zahlungsarten 4 oder 6 verwenden.<br><br>Siehe hierzu die Doku. |
| 3.00 | 04.02.2021 | - Die Struktur CamtReadStruct wurde im Reservebereich um die Felder BatchBetrag und NachrichtMsgId ergänzt (siehe Doku). Für Bestandsanwender besteht nur Handlungsbedarf, wenn diese Felder verwendet werden sollen.<br><br>- Die Funktion SepaTools_GetBLZKonto liefert nun auch Ergebnisse für polnische Bankverbindungen.<br><br>- In die Datenbanken wurden die Purpose Codes und Category Purpose Codes aus den ISO External Code Sets integriert. Für diesen Zweck stehen die neuen Funktionen SepaTools_GetPurposeCode und SepaTools_SearchPurposeCodes zur Verfügung. |
| 2.91 | 03.11.2020 | - Die Struktur SepaTools_CreateExt wurde geändert (ein neues Feld). |
| 2.90 | 04.08.2020 | - Es gibt vier neue Funktionen zur Erzeugung von AZV-Dateien im XML-Format nach der ISO-Spezifikation pain.001.001.09. Details entnehmen Sie bitte der Dokumentation.<br><br>- Zur Prüfung der IBAN gibt es die neue Funktion SepaTools_CheckIBAN_Strong. Diese Funktion filtert keine ungültigen Zeichen aus der IBAN aus, sondern liefert einen negativen Rückgabewert (zu Details siehe bitte die Doku).<br><br>- Innerhalb der Struktur ***XMLWriteExtStruct*** gibt es das zusätzliche Feld ***StructPrtry***. Der dadurch erforderliche Speicherplatz wird dem Feld ***Reserve1*** entnommen. |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | Für Bestandsanwender ändert sich dadurch nichts. |
| 2.81.0 | 29.04.2020 | - Bei der Erzeugung von CGI-Dateien wurde ein Speicherfehler bereinigt. |
| 2.80.0 | 10.02.2020 | - Bei der Konvertierung von MT940-Dateien wurde eine geringfügige Programmschwäche bereinigt.<br><br>- Es gab ein Problem bei der Erstellung von Schweizer ESR-Lastschriften. Dieses wurde bereinigt. |
| 2.71.0 | 02.11.2019 | - Beim Einlesen von Camt-Dateien in eine Struktur wurden bei langen Texten für das Feld „ZusatzInfo“ einige nachfolgende Felder überschrieben. Das Problem ist bereinigt.<br><br>- Die Struktur ***CamtReadStruct*** wurde im Reservebereich um das Feld ***Whg*** für die Währung des Umsatzes ergänzt. |
| 2.70.0 | 07.09.2019 | - Für die Funktion ***SepaTools_SetVersionUndLand*** steht auch der Wert 7 für die XML-Version 3.3 zur Verfügung.<br><br>- Die Funktion ***SepaTools_GetBLZKonto*** liefert jetzt zusätzlich auch Ergebnisse für die Schweiz, Liechtenstein und Niederlande.<br><br>Aufgrund dieser Erweiterung musste die Struktur ***BLZKontoStruct*** geändert werden. Die Änderung erfolgte aber im Reservebereich, so dass für die bisherigen Nutzer keine Änderung erforderlich ist. Bitte beachten Sie hierzu auch die Dokumentation.<br><br>- Die Struktur ***XMLOptionStruct*** wurde im Reservebereich um zwei zusätzliche Einträge ergänzt. Siehe hierzu die Doku. |
| 2.60.2 | 03.05.2019 | - Bei der Funktion SepaTools_GetBLZKonto ist der Rückgabewert -9 ersatzlos entfallen.<br>- Die Datenbanken müssen/sollen ausgetauscht werden. |
| 2.60 | 05.02.2019 | - Die IBAN-Regel 049 wurde geändert. |
| 2.50 | 14.08.2018 | - Die Funktionalität des Feldes ***EilUebw*** in der Struktur ***XMLOptionStruct*** wurde erweitert.<br><br>- In der Struktur ***XMLOptionStruct*** gibt es ein zusätzliches Feld ***ISOSonder***.<br><br>- Im Reservebereich der Struktur ***XMLWriteStructExt*** gibt es nun das zusätzliche Feld ***AusfZeit*** (Instant-Zahlung mit ISO2017).<br><br>- Innerhalb der ***CamtReadstruct*** wurde im Reservebereich das Feld ***RvslInd*** (Storno) eingefügt. |
| 2.40 | 06.05.2018 | - Die Struktur ***CGICreateStruct*** wurde im Reservebereich um die Felder ***ShortMsgId*** und ***WhgSort*** erweitert.<br><br>- Die Struktur ***XMLWriteExtStruct*** wurde um zwei Felder für den Schweizer Zahlungsverkehr erweitert. |
| 2.30 | 01.02.2018 | - Die Funktion ***SepaTools_ConvertMT940*** wurde um den Paramter ***Compress*** erweitert. Damit wird die Komprimierung in der ZIP-Datei gesteuert. Näheres entnehmen Sie bitte der Dokumentation. |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | - Die von der Funktion ***SepaTools_ConvertCamt54ToDTI*** optional nutzbare Möglichkeit einer Mappingtabelle wurde erweitert. Näheres siehe Dokumentation. |
| 2.21 | 22.11.2017 | - Es gibt die neue Funktion ***Sepatools_ConvertCamt53ToMT940***. Damit können Camt.053-Dateien in MT940-Dateien konvertiert werden. |
| 2.20 | 31.10.2017 | - Mit der API können pain.007-Dateien erzeugt werden. Diese Dateiart dient zum Rückruf von Lastschriften (z.B. wegen Doppeleinreichung). Diese Funktionalität wird grundsätzlich ab November 2017 unterstützt.<br><br>Die Unterstützung von pain.007-Dateien durch die Zahlungsdienstleister ist optional. Im Gegensatz zur Verarbeitung von XML-Zahlungsverkehrsdateien gibt es für diese Dateiart keine Verpflichtung der Banken.<br><br>- Mit der API können CGI-Dateien erstellt werden. Dadurch ist es möglich Auslandszahlungen außerhalb des SEPA-Raums im XML-Formt darzustellen. Zu berücksichtigen ist aber, dass dieses Format derzeit erst von wenigen Banken (meist Großbanken) unterstützt wird.<br><br>Die DLL-Funktionen für dieses Format wurden mit der NORDEA Bank in Finnland erfolgreich verifiziert.<br><br>- Die Datenstruktur ***CamtReadStruct*** wurde im Reservebereich um das Feld ***ZusatzInfoTx*** erweitert. In diesem Feld können zusätzliche Informationen auf Dtls-Ebene (AddtlTxInf) abgelegt werden. Das Feld wird teileweise von Schweizer Banken für die ESR-Nummer verwendet.<br><br>Auch über die Funktion ***SepaTools_ConvertCamtToCSV*** kann dieses Feld ausgewertet werden.<br><br>- Bei Schweizer TA-Lastschriften (Zahlungsart 102) gibt es eine Erweiterung bzgl. der ESR-Referenznummer. Die Struktur ***XMLWriteExtStruct*** wurde um das Feld ***ChLSCreditorESR*** erweitert. |
| 2.10 | 02.08.2017 | - Die Struktur ***CamtDTIStruct*** wurde im Reservebereich erweitert.<br><br>- Es gibt die neue Funktion ***CreateXMLExt***. Diese behebt einen „Geburtsfehler“ der Funktion ***CreateXML*** (fehlender Reservebereich).<br><br>- Der Block ***PmtTpInf*** kann auf Transaktionsebene gelegt werden. Siehe hierzu die Funktion ***CreateXMLExt***.<br><br>- Die Struktur ***XMLWriteExtStruct*** wurde im Reservebereich erweitert. Für den Zahlungsempfänger und den Auftraggeber können Adressdaten übergeben werden. Ebenso ist die Angabe des „Category Purpose“-Codes möglich. |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | Es gibt das zusätzliche Feld ***ChAccountPrtry***.<br><br>- Auch bei der Funktion ***SepaTools_ConvertCSVtoXML*** können für den Auftraggeber und den Zahlungsempfänger Adressdaten übergeben werden. Die Struktur ***CSVConvertStruct*** wurde geändert. Näheres finden Sie in der Dokumentation<br><br>- Bei Verwendung der Funktion ***SepaTools_ConvertDTAZVtoXML*** werden ebenfalls die Adressdaten von Auftraggeber und Empfänger in die XML-Datei geschrieben.<br><br>- Die Sortierung der Datensätze nach Währung ist möglich. Dies ergibt aber nur bei Zahlungen für den Schweizer Zahlungsverkehr einen Sinn, da ansonsten die Währung immer EUR ist.<br><br>- Ebenso nur für die Einreichung der Zahlungen in der Schweiz können in der neuen Struktur ***XMLCreateExtStruct*** Kontaktinformationen für den GroupHeader hinterlegt werden.<br><br>Bitte beachten Sie hierzu die Funktion ***CreateXMLExt***.<br><br>- Der Paramter ***Netz*** in der Funktion ***SepaTools_Init*** hat keine Bedeutung mehr. Die Datenbanken arbeiten jetzt immer im Netzwerkmodus. |
| 2.00 | 07.05.2017 | Ab der DK-Version 3.0 (gültig ab November 2016) gibt es im Zeichensatz Beschränkungen bzgl. der Schrägstriche (Slash). Über optionale Einstellungen (siehe bei den verschiedenen Versionen) können Sie entscheiden, ob Sie eine automatische Anpassung wünschen oder nicht.<br><br>In der Funktion SepaTools_SetOptions gibt es für die Behandlung der Schrägstriche den neuen Feldinhalt „CheckSlash“. Die Funktionsweise ist in der API-Dokumentation beschrieben.<br><br>Das Prüfziffernverfahren B1 wurde geändert.<br><br>Es gibt ein neues Prüfziffernverfahren E4.<br><br>Die IBAN-Regel 54 ist entfallen.<br><br>Bei der Konvertierung von MT940-Dateien nach Camt.053 wurde der Ergänzungstextschlüssel nicht korrekt ausgewertet. Der Fehler wurde bereinigt. Ebenso wurden Anpassungen an das in den Niederlanden verwendete MT940-Format vorgenommen.<br><br>Beim Einlesen von Camt-Auszügen wurden Besonderheiten der Schweizer Postfinance berücksichtigt.<br><br>Der aktuelle Bankleitzahlenbestand der Deutschen Bundesbank ist in allen Datenbanken enthalten. Er hat eine Gültigkeit vom 06.06.2017 bis zum 03.09.2017.<br><br>Die Datenbank, der an SEPA teilnehmenden Banken wurde aktualisiert. Aktualisierungsstand: 06.05.2017. |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| 1.90.0 | 03.02.2017 | Im Camt-Modul können Camt.054-Dateien (Sammlerauflösungen) in DTI-Dateien konvertiert werden.<br><br>Das neue Prüfziffernverfahren E3 wurde integriert. |
| 1.80.1 | 08.11.2016 | Beim Einlesen von Camt-Nachrichten werden auch die Bank Transaction Codes nach ISO ausgewertet. Die zusätzlichen Datenfelder wurden im Reservebereich untergebracht, so dass keine Änderungen erforderlich sind, wenn Sie diese Felder nicht nutzen wollen. |
| 1.80.0 | 02.11.2016 | Der aktuelle Bankleitzahlenbestand der Deutschen Bundesbank ist in allen Datenbanken enthalten. Er hat eine Gültigkeit vom 05.12.2016 bis zum 05.03.2017.<br><br>Die Datenbank, der an SEPA teilnehmenden Banken wurde aktualisiert. Aktualisierungsstand: 02.11.2016.<br><br>Das Prüfziffernverfahren 74 wurde erweitert.<br><br>Die IBAN-Regel Nr. 42 (Deutsche Bundesbank) wurde erweitert. Ein zusätzlicher Kontokreis für 10stellige Kontonummern wurde für die IBAN-Berechnung freigegeben.<br><br>Bei der Funktion SepaTools_CheckPruefziffer gibt es denn zusätzlichen Rückgabecode -5, wenn keine Kontonummer oder die Kontonummer 0 übergeben wurde.<br><br>Beim Einlesen von Camt-Formaten werden zusätzliche Felder für den Ursprungsbetrag bei Lastschriftrückgaben und die anfallenden Gebühren bereitgestellt. Die zusätzlichen Felder sind im Reservebereich untergebracht, so dass bei der Nichtnutzung keine Strukturen geändert werden müssen. |
| 1.70.0 | 02.08.2016 | Zusätzliche Funktionen zur Erzeugung des Deutschen Auslandszahlungsverkehr (DK-Spezifikation) stehen zu Verfügung. |
| 1.60.2 | 18.05.2016 | Eine Fehlerbereinigung aus der Änderung der Version 1.60.1 vom 17.05.2016 |
| 1.60.1 | 17.05.2016 | In der Struktur „CamtReadStruct“ wurden die Felder Cred_CI und Debt_CI ergänzt. Der Reservebereich verkleinert sich damit auf 928 Byte. |
| 1.60.0 | 09.05.2016 | - Es wird die ab dem 20. November 2016 gültige DK-Version 3.0 (neues XML-Schema) unterstützt. Dabei gibt es folgende wesentliche Neuerungen:<br>  - Es gibt nur noch Eil-Lastschriften (D+1).<br>  - Das Instrument RCUR gilt nun für Erst- und Folgelastschriften.<br>  - Bei Mandatsreferenzen können Leerzeichen enthalten sein.<br>  - Der schematische Aufbau für Mandatsänderungen wurde geändert.<br><br>- Zusätzliche Informationen sind in einem Dokument im Downloadbereich von www.sepa-tools.de hinterlegt. |
| 1.50.0 | 01.11.2015 | - Die Einreichung von XML-Dateien in der Schweiz wird nun unterstützt.<br><br>Aufgrund der Unterstützung der XML-Erzeugung für die Schweiz hat |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | sich die Datenstruktur geändert, bzw. wurde ergänzt. Aufgrund der differenzierteren Steuerung des Zahlungsverkehrs in der Schweiz werden für die Unterstützung dieser Funktionalität mehr Datenfelder benötigt.<br><br>Wenn die neuen Felder nicht verwendet werden sollen, so müssen Sie nur die Felder ***StructZweck***, ***StructTyp*** und ***BatchBooking*** anpassen.<br><br>Die zusätzlichen Felder wurden im Reserve-Bereich untergebracht. Die gesamte Größe der Struktur hat sich nicht verändert. |
| 1.49.0 | 18.08.2015 | - In der Funktion *SepaTools_GetVersion* wurde ein Fehler beseitigt (Integer für Version war 0 anstatt 1).<br><br>- Das Log-File gab bei der Funktion „SepaTools_CamtOpen“ den Wert „SepaTools_CamtRead“ aus. Dieser Fehler wurde beseitigt. |
| 1.48.0 | 05.05.2015 | - Im Rahmen des Camt-Moduls können MT940-Dateien in Camt.053-Dateien konvertiert werden. |
| 1.47.0 | 07.02.2015 | - Die Bedeutung des Parameters *Skip* in der Funktion *SepaTools_GetTargetDatum* wurde erweitert.<br><br>- Die Funktion *SepaTools_SetLogFile wurde erweitert*. Zusätzlich zur Textdatei können auch Meldungsfenster ausgegeben werden.<br><br>- Es gibt die zusätzliche Funktion *SepaTools_SetLogFileInhalt*.<br><br>- Innerhalb der SEPA-XML-Dateien können Umlaute weitergegeben werden. Die Steuerung erfolgt über die Funktion *SepaTools_SetOptions*. Die Struktur der Funktion *SepaTools_SetOptions* wurde dahingehend modifiziert. |
| 1.46.0 | 17.11.2014 | - Es gibt die neue Funktion *SepaTools_SetLogFile*. Damit kann der Ablauf der DLL-Funktionen in einer Textdatei mitgeschrieben werden.<br><br>- Die Bedeutung des Rückgabecodes -6 bei der Funktion *SepaTools_CheckCI* wurde geändert.<br><br>- Die Struktur *BLZKontoStruct* wurde erweitert.<br>- Die Struktur *DTAInfoStruct* wurde erweitert. |
| 1.45.3 | 03.10.2014 | - Im Rahmen des Camt-Moduls können nun auch DTA/DTI-Dateien in Camt.054-Dateien konvertiert werden. Dazu gibt es die neue Funktion *SepaTools_ConvertDTI*.<br><br>- Im Rahmen des Camt-Moduls können nun auch Pain.002-Dateien ausgewertet werden.<br><br>- Die Struktur *CamtReadStruct* wurde im unteren Bereich geändert. In den Reservefeldern wurden Felder für die Auswertung von Pain.002-Dateien aufgenommen.<br><br>- Die Struktur *CamtOpenStruct* wurde erweitert. |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | - Die Struktur *CamtCSVStruct* wurde erweitert. |
| 1.45.0 | 27.08.2014 | - Für die Funktion *CheckCI* gibt es den zusätzlichen Rückgabecode -6.<br><br>- Die Struktur *CamtReadStruct* wurde erweitert.<br><br>- Die Struktur *XMLOptionStruct* wurde erweitert. Über den Wert *CompressXML* kann definiert werden, dass die XML-Datei komprimiert (ohne Zeilenschaltungen) ausgegeben wird.<br><br>- Die Struktur XMLWriteExtStruct wurde erweitert. Der Wert *BatchBooking* (BtchBookg) kann explizit gesetzt werden.<br><br>- Es gibt für Spezialanwendungen eine neue Datenbank Sepa_Big (Sepa_Big.dat und Sepa_Big.idx). In dieser Datenbank wurden die unter SEPA erreichbaren deutschen Banken mit den Ortsbezeichnungen und den Filialen aus dem Bankleitzahlenbestand angereichert.<br><br>- Diese neue Datenbank ergibt nur im Zusammenhang mit der Funktion S*epaTools_GetSEPABank* einen Sinn, da hier die Banken nach dem Ortsnamen gesucht werden. Bei der Verwendung der neuen Datenbank sind die Dateien Sepa_Big.dat und Sepa_Big.idx in Sepa.dat und Sepa.idx umzubenennen.<br><br>- In diesem Zusammenhang sollten Sie auch die Erweiterung der Funktion *SepaTools_SetOptions*, hier den neuen Wert *NotNameOrt* beachten. |
| 1.44.3 | 10.07.2014 | - Die Struktur *XMLOptionStruct* wurde erweitert.<br><br>- Die Struktur *XMLWriteExtStruct* wurde erweitert.<br><br>- Beim Konvertieren von Camt-Dateien kann jetzt auch der strukturierte Verwendungszweck ausgelesen werden.<br><br>Bitte beachten Sie hierzu die Änderungen am Ende der Struktur *CamtReadStruct*. |
| 1.44.0 | 08.05.2014 | - Bei der direkten Konvertierung von DTA-Dateien gibt es einen zusätzlichen Platzhalter (***I*LS**) für Mandatsdaten.<br><br>- Für die Funktion *CheckCI* gibt es den zusätzlichen Rückgabecode -6.<br><br>- Die Struktur CamtReadStruct wurde erweitert.<br><br>- Bei der Funktion *XMLLesen* wurde das Feld SammlerSumme in der Struktur *XMLReadStruct* nicht gefüllt. Dieser Fehler ist beseitigt. |
| 1.43.1 | 20.02.2014 | - Die Datei SepaUnZIP.exe wird zum Entpacken der Camt-Dateien nicht mehr benötigt. Die Datei DUNZIP.DLL muss aber vorhanden sein.<br><br>- Die Strukturen *CamtCSVStruct* und *CamtOpenStruct* wurden erwei- |

| Version | Datum | Dokumentänderungen |
|---|---|---|
| | | tert.<br><br>- Für die Funktion *SetOptions* gibt des die zusätzlichen Variablen „SammlerTrennen“ und „MaxXMLSatz". |
| 1.43 | 05.02.2014 | - Es gibt die zusätzliche Funktion *GetBICInfo*. Bitte die Dokumentation beachten<br><br>- Es stehen zusätzliche Funktionen zum Auslesen von Camt-Dateien zur Verfügung.<br><br>- Über die Funktion *GetBankInfo* kann nun auch die COR1-Teilnahme geprüft werden.<br><br>- Es gibt den neuen Rückgabecode -123. Dieser wird ausgegeben, wenn in der Funktion *XMLWrite* der B2B auf 2 (COR1) gesetzt wird und die Bank des Empfängers dies nicht unterstützt.<br><br>- Bei der Konvertierung von CSV-Dateien in XML-Dateien kann optional eine EndToEnd-ID übergeben werden. Diese überschreibt die sonst automatisch ermittelte EndToEnd-ID.<br><br>- Die Funktion *SetOptions* wurde erweitert.<br><br>- Bei der Funktion SepaTools_GetBankInfo gibt es einen zusätzlichen, positiven Rückgabecode (ein Hinweis, kein Fehler). |
| 1.42 | 06.11.2013 | - Der Rückgabecode -115 bei der Funktion WriteXML ist entfallen. Es können auch XML-Dateien ohne Verwendungszweck erstellt werden. |
| 1.41 | 31.08.2013 | - Für die Funktionen *XMLWrite* und *XMLWriteExt* gibt es die zusätzlichen Ergebniscodes -120 bis -122.<br><br>- Die Deutsche Bank hat ihre IBAN-Regel geändert. Betroffen sind 7stellige und 10stellige Kontonummern. |
| 1.40 | 04.08.2013 | - Der Reserve-Bereich der Struktur XMLWriteExtStruct wurde von 500 Byte auf 1.500 Byte erweitert. Dies dient künftigen Erweiterungen.<br>- Bei der Funktion *CloseXML* gibt es den zusätzlichen Fehlercode -8. |
| 1.39 | 29.06.2013 | - Erweiterung der Funktion *SetOptions*<br>- Neue, erweiterte Funktion *WriteXMLExt*<br>- Neue Funktion *ConvertDTAZVtoXML*<br>- Neue Funktion *ConvertCSVtoXML*<br>- Neue Funktion *GetBLZKonto*<br>- Optionale Erweiterung des Parameters B2B in den Strukturen *XMLWriteStruct, DTAInfoStruct* und *XMLReadStruct.*<br>- Bei den Funktionen *CheckSEPATeilnahme* und *GetBankInfo* gibt es den zusätzlichen Rückgabecode -3.<br>- Bei der Funktion GetBIC gibt es einen zusätzlichen Rückgabecode -2. |

# 2 Grundsätzliches

Die folgende Dokumentation beschreibt eine Bibliothek, die zur Unterstützung von Fremdprodukten bei der SEPA-Implementierung eingesetzt werden kann. Es handelt sich dabei um eine Windows-DLLs für 32-Bit Windows-Programme.

Das Tool wurde grundsätzlich für die deutsche Kreditwirtschaft entwickelt. In weiten Teilen werden aber auch österreichische Bankverbindungen unterstützt. Aufgrund der Angabe einer 5stelligen Bankleitzahl, wird erkannt ob es sich um eine österreichische Bankverbindung handelt.

Der Name der DLL lautet **SepaTools.dll**.

Alle Daten werden in definierten Datenstrukturen übergeben, bzw. zurückgegeben. Dabei wird der Syntax der Sprache **C** verwendet.

## *2.1 Datenstrukturen*

Die API-Funktionen erwarten die angegebenen Datenstrukturen mit mindestens der angegebenen Größe. Auf alle Strukturen wird mittels Pointer zugegriffen.

Bei char-Variablen ist als Länge immer die tatsächliche maximale Länge+1 für die abschließende NULL angegeben.

**Hinweis:**

**Es ist Aufgabe der aufrufenden Applikation, dass die übergebenen Speicherbereiche groß genug sind.**

Strukturen, die gefüllt werden, werden bis zur Größe der übergebenen Struktur gefüllt. Es liegt in der Verantwortung des aufrufenden Programms, dass die Speicher bereits entsprechend allokiert sind.

**Hinweis:**

**Einige Datenstrukturen enthalten Reservebereiche. Bitte achten Sie darauf, dass diese mit binären Nullen (NULL) initialisiert werden, da Sie sonst unerwartete (fehlerhafte) Ergebnisse erhalten.**

Alle int-Werte sind 4 Byte groß. Auf double-Werte für Betragsangaben wurde verzichtet, da es bei Visual Basic Probleme mit der Byte-Ausrichtung in Records gibt. Anstelle dessen werden die Betragsangaben als Zeichenkette in Cent (ohne Nachkommastellen) übergeben.

## *2.2 Returncodes*

Alle Returncodes werden in Regelfall als negative ***int*** –Werte (32-Bit / 4 Byte) zurückgegeben. Werden andere Returncodes zurückgegeben, so ist dies explizit angegeben.

Bei einem Returncode von 0 ist alles in Ordnung. Werden positive Returncodes zurückgegeben, so stellen diese lediglich Hinweise dar. Die Anwendung kann aber genauso fortgesetzt werden, als wäre der Returncode 0 zurückgegeben worden.

## *2.3 DLL-Deklaration*

Alle DLL-Funktionen sind ***stdcall*** deklariert. Die Groß-/Kleinschreibung der Funktionsaufrufe ist zu beachten. Strukturen sind standardmäßig mit einer Ausrichtung von 8 Byte dargestellt (#pragma pack (8)).

In Praxistests hat sich herausgestellt, dass u.U. nicht alle Compiler (teilweise Cobol Compiler) mit der Ausrichtung von 8 Byte bei Datenstrukturen umgehen können. Um diesem Problem Rechnung zu tragen, wird die DLL zusätzlich mit einer Byte Ausrichtung von 1 Byte (#pragma pack (1)) zur Verfügung gestellt.

Der Name dieser DLL lautet dann **SepaTools_P1.dll**. Gegebenenfalls muss der Dateiname umbenannt werden.

Nachdem manche Compiler auch eine Ausrichtung von 4 Byte verwenden, gibt es zusätzlich noch die DLL mit dem Namen **SepaTools_P4.dll** (#pragma pack 4). Gegebenenfalls muss der Dateiname umbenannt werden.

Die API ist in der Programmiersprache Delphi erstellt. Es gibt daher keine C-Header Dateien. Diese müssen Sie selbst erstellen.

**Wir empfehlen die DLL dynamisch über die Windows-Funktion LoadLibrary zu laden und über die Funktion GetProcAddress die Funktionspointer zuzuweisen.**

## *2.4 Debug-Version*

Die Debug-Versionen entfallen ab der Version 1.47.0. Anstelle dessen kann die Funktion SepaTools_SetLogFile verwendet werden.

## *2.5 Datenbanken*

Um vielfältige Fehlerprüfungen vornehmen zu können, werden von SepaTools zwei Datenbanken benötigt. Es handelt sich hier um den von der Deutschen Bundesbank herausgegebenen Bankleitzahlenbestand (BLZ.dat und BLZ.idx), sowie um das Verzeichnis der unter SEPA erreichbaren Banken mit zusätzlichen Informationen (Sepa.dat und Sepa.idx).

Grundsätzlich werden diese Dateien (Bestandteil der API) im aktuellen Verzeichnis erwartet. Sie können allerdings über die Funktion SepaTools_Init einen abweichenden Pfad für diese Datenbanken übergeben.

Diese Datenbanken werden regelmäßig (mindestens vierteljährlich) über die Website http://www.Sepa-Tool.de aktualisiert. Dies sollte auch von der Applikation erfolgen.

Ab der aktuellen Version wird geprüft, ob die Datenbanken (Sepa.dat und Sepa.idx) zur aktuellen Programmversion passen. Ist dies nicht der Fall, so wird dies über die Funktion SepaTools_Init mit einem entsprechenden Fehlercode zurückgemeldet.

Bei Programmaktualisierungen über die Webseite http://www.Sepa-Tool.de sollten daher immer die DLL und die Datenbanken gemeinsam aktualisiert werden.

Die Datenbanken arbeiten nun immer im Netzwerkmodus.

## *2.6 Datenbank_Mini*

Die oben beschriebenen Datenbanken Sepa.dat und Sepa.idx sind sehr groß. Grund dafür ist, dass in diesen Datenbanken die BICs und Adressen aller in Europa über SEPA erreichbaren Kreditinstitute enthalten sind.

Diese Informationen sind aber nicht für alle Funktionen der API erforderlich. Wenn Sie nur die nachfolgend aufgeführten Funktionen verwenden, so können Sie auch die wesentlich kleineren Dateien Sepa_Mini.dat und Sepa_Mini.idx verwenden. Diese Datenbank enthält die vorstehend genannten BICs nicht.

Die Dateien Sepa_Mini.dat und Sepa_Mini.idx sind dabei in Sepa.dat und Sepa.idx umzubenennen.

Die Dateien BLZ.dat und BLZ.idx sind wie bisher zu verwenden.

## *2.7 Funktionen für Datenbank_Mini*

Für folgende Funktionen ist es ausreichend, wenn die Dateien Sepa_mini.dat und Sepa_Mini.idx (entsprechend umbenannt) verwendet werden.

SepaTools_Version
SepaTools_Init
SepaTools_Free
SepaTools_Info
SepaTools_GetBIC
SepaTools_CheckIBAN
SepaTools_CheckIBANLand
SepaTools_CheckCI
SepaTools_ConvertBLZKonto
SepaTools_SetErfahrung
SepaTools_GetIBANLand
SepaTools_GetBLZ
SepaTools_CheckPruefziffer

Die Funktionen xxUserBIC und xx UserIBAN können grundsätzlich auch verwendet werden, ergibt aber keinen Sinn ohne die XML-Erstellung.

## *2.8 Datenbank_Big (Datenbank_Big.zip)*

Die Datenbank Sepa.dat/Sepa.idx enthält alle über SEPA erreichbaren Banken. Grundlage dafür ist das „SCT register of participants“ des EPCs und das „SCL-Directory“ der Deutschen Bundesbank. Im SCL-Directory sind alle erreichbaren BICs enthalten.

Leider bietet dieses Verzeichnis keine Ortseinträge, so dass die Suche nach einer bestimmten Bank über die Funktion *SepaTools_GetSEPABank* häufig nicht zu dem gewünschten Ergebnis führt (Ort nicht verfügbar).

Um hier zumindest für die deutschen Banken ein besseres Ergebnis zu erzielen, wurde die Datenbank_Big (bestehend aus den Dateien Sepa_Big.dat und Sepa_Big.idx) mit den Ortseinträgen aus dem deutschen Bankleitzahlenverzeichnis angereichert.

Diese zusätzliche Datenbank ergibt nur in Verbindung mit der Funktion *SepaTools_GetSEPABank* einen Sinn. Soll diese Datenbank verwendet werden, so sind die Dateien Sepa_Big.dat und Sepa_Big.idx in Sepa.dat und Sepa.idx umzubenennen.

Bitte beachten Sie in diesem Zusammenhang auch die Variable *NotNameOrt* in der Struktur *XMLOptionStruct*.

## *2.9 Umlaute in SEPA XML-Dateien*

Grundsätzlich ist es erlaubt, dass in SEPA XML-Dateien auch Umlaute weitergegen werden können. Die Kreditinstitute sind verpflichtet, SEPA-Dateien mit Umlauten anzunehmen. Allerdings können die Kreditinstitute auf der Empfängerseite die Umlaute durch andere Zeichen oder durch Leerzeichen ersetzten.

Wenn Sie die entsprechende Option (SepaTools_SetOptions) daher wählen, kann trotzdem nicht garantiert werden, dass die Umlaute auch den Empfänger der Zahlung erreichen. Im Zweifelsfall sollten Sie den Einsatz der Umlaute daher mit den betroffenen Banken abstimmen.

Wenn Sie keine besonderen Maßnahmen ergreifen, so werden innerhalb von SepaTools die Umlaute ÄÖÜäöüß in Ae, Oe, Ue, ae, oe, ue, ss dargestellt.

# 3 Abweichungen und Hinweise für die 64-Bit Version

Die Anwendung der 64-Bit Version ist grundsätzlich identisch mit der Anwendung der 32-Bit Version. Es gibt einige kleine Unterschiede und Hinweise, die an dieser Stelle beschrieben werden.

Wenn an dieser Stelle keine Hinweise erfolgen, so gilt die Dokumentation der 32-Bit Version.

## *3.1 Dateinamen*

Der Dateiname für die 64-Bit Version der DLL lautet ***SepaTools64.dll*** für die DLL mit einer Byte-Ausrichtung von ***8 Byte*** und ***SepaTools64_P1.dll*** für die DLL mit einer Byte-Ausrichtung von **1 Byte**.

## *3.2 Datenstrukturen*

Die Datenstrukturen der 64-Bit Version sind identisch mit denen der 32-Bit Version. Das bedeutet, dass alle Integer-Werte (int) 4 Byte groß sind. Alle Zeichen, Zeichenketten und Zeiger (Pointer) auf Zeichenketten enthalten Zeichen mit einer Größe von 1 Byte. Das ist identisch mit der 32-Bit Version.

**Bitte beachten:**

**Es gibt keine Wide Char und Wide Strings, bei denen ein Zeichen aus zwei Byte (Word) besteht. Es sind nur AnsiChar und Pointer (PAnsiChar) darauf zulässig.**

## *3.3 Einbindung der DLL*

Bei der DLL handelt es sich um **kein** COM-Objekt. Die Einbindung erfolgt ebenso wie bei der 32-Bit Version über die Funktion ***LoadLibrary***. Die Funktions-Pointer werden über die Funktion ***GetProcAddress*** abgerufen.

Für die Camt-Funktionalitäten werden die DLLs DUnZip32.dll und DZip32.dll **nicht** mehr benötigt.

## *3.4 Datenbanken*

Es werden die gleichen kompakten Datenbanken wie bei der 32-Bit Version verwendet. Diese sind aber nicht direkt in die 64-Bit DLL einbindbar.

Die Anbindung erfolgt über einen 32-Bit Datenbank-Server mit der Bezeichnung SepaDB.exe. Dieser wird von der 64-Bit DLL intern angesprochen und administriert.

## *3.5 SepaTools_Init*

Dies ist der einzige Funktionsaufruf, der von den Parametern und Rückgabewerten in Teilen von der 32-Bit Version abweicht.

### *3.6 Funktionsaufruf*

**int SepaTools_Init(const char *DataPath,**
**const char *TempPath,**
**const char *UserPath,**
**const char *Lizenz,**
**const char *Serverpath)**

### *3.7 Parameter*

| | |
|---|---|
| ***DataPath*** | Siehe 32-Bit Version |
| ***TempPath*** | Siehe 32-Bit Version. |
| ***UserPath*** | Siehe 32-Bit Version. |
| ***Lizenz*** | Siehe 32-Bit Version.<br><br>Sie benötigen für die 64-Bit Version einen eigenen, von der 32-Bit Version abweichenden Lizenzschlüssel. |
| ***ServerPath*** | Übergenen Sie bitte hier den Pfad, in dem sich der Datenbank-Server SepaDB.exe befindet.<br><br>Bitte übergeben Sie nur den Pfad und keinen Dateinamen (Programmnamen). |
| ***Netz*** | Dieser Parameter existiert in der 64-Bit Version nicht mehr. Die Datenbanken werden **immer** im Netzwerkmodus geöffnet. |

### *3.8 Returncodes*

Es gibt folgende, zusätzliche Returncodes für diese Funktion:

| | |
|---|---|
| -20 | Der Datenbank-Server wurde nicht gefunden. Bitte überprüfen Sie den übergebenen Pfad in der Variablen ***ServerPath***. |
| -21 | Der Datenbank-Server und die 64-Bit DLL passen nicht zusammen. Bitte besorgen Sie sich von der Webseite www.sepa-tool.de die aktuellsten Versionen. |
| -998 | Der Datenbank-Server ist nicht aktiv. Dies kann eigentlich nur vorkommen, wenn der Datenbank-Server durch einen Programmabsturz (was eigentlich nicht vorkommen sollte) oder manuell beendet wurde.<br><br>**Bitte beachten:**<br><br>**Dieser Returncode kann grundsätzlich bei allen DLL-Funktionen vorkommen, da vor jedem Funktionsaufruf geprüft wird, ob der Datenbank-Server aktiv ist.** |

## *3.9 Versionsführung*

Die 64-Bit Version erhält immer die gleiche Versionsnummer wie die 32-Bit Version. Das basiert auf dem immer fachlich gleichen Inhalt der beiden Versionen.

Das Versionsdatum kann u.U. abweichen, da die beiden Versionen nicht zwingend am gleichen Tag erstellt werden.

## *3.10 Performance-Tests*

Mit der 64-Bit Version wurden Performance-Tests auf zwei Notebooks durchgeführt. Dabei wurden über die Funktion SepaTools_WriteXMLExt unterschiedliche Anzahlen von Lastschriften erzeugt.

Da in allen Fällen BIC und IBAN aus BLZ und Kontonummer ermittelt wurden, erfolgte auch in allen Fällen ein Zugriff auf den Datenbank-Server. Für die Tests wurden folgende Notebooks verwendet.

Bereits im Vorfeld ist festzuhalten, dass die Ergebnisse des Tests sehr stark von der Leistungsfähigkeit der Festplatte abhängig sind. Ebenso spielt die sonstige Auslastung des PCs eine entscheidende Rolle. Die angegebenen Werte sind damit nur Anhaltswerte.

| | *Test1* | *Test2* |
|---|---|---|
| Modell | Lenovo ThinkPad T530 | Lenovo ThinkPad P140 p |
| Baujahr | Juni 2013 | November 2014 |
| Festplatte | 500 GB | 1 TB |
| Betriebssystem | Windows 7 SP 1 | Windows 8.1 Pro |
| Hauptspeicher | 8 GB | 16 GB |
| Prozessor | I7-3740 QM CPU 2,7 GHz | I7-4910 QM CPU 2,9 GHz |

**Ergebnisse der Tests:**

| *Anzahl Lastschr.* | *Test1 Zeit MM:SS* | *Test2 Zeit MM:SS* | *Dateigröße* | *Anzahl Zeilen in der XML-Datei* |
|---|---|---|---|---|
| 3.000 | 00:04 | 00:10 | 2,3 MB | 84.099 |
| 10.000 | 00:13 | 00:15 | 7,5 MB | 280.071 |
| 20.000 | 00:26 | 00:33 | 15,1 MB | 560.043 |
| 50.000 | 01:12 | 01:17 | 37,7 MB | 1.400.043 |
| 100.000 | 02:34 | 02:35 | 75,5 MB | 2.800.071 |
| 150.000 | 03:36 | 03.59 | 113 MB | 4.200.099 |
| 200.000 | 04:47 | 05:26 | 151 MB | 5.600.043 |
| 250.000 | 05:56 | 07:03 | 189 MB | 7.000.071 |
| 300.000 | 07:27 | 08:08 | 227 MB | 8.400.099 |
| 350.000 | 08:59 | 09:27 | 264 MB | 9.800.044 |
| 400.000 | 11:36 | 11:41 | 302 MB | 11.200.071 |
| 450.000 | 12:24 | 13:57 | 340 MB | 12.600.099 |
| 500.000 | 13:35 | 16:36 | 378 MB | 14.000.043 |

# 4 Bankfachliche Grundlagen

Die DLL stellt eine technische Schnittstelle zur Realisierung von SEPA-Funktionalitäten dar. Die bankfachlichen Grundlagen werden vorausgesetzt. Trotzdem die Erklärung wesentlicher Begriffe und Hinweise auf Informationsquellen.

| ***Begriff*** | ***Erklärung*** |
|---|---|
| BLZ - Bankleitzahl | Die nationale Adresse einer Bank. Die BLZ ist in Deutschland 8stellig und in Österreich 5stellig. Informationen zur deutschen Bankleitzahl erhalten Sie über den Link:<br><br>http://www.bundesbank.de/Navigation/DE/Kerngeschaeftsfelder/Unbarer_Zahlungsverkehr/Bankleitzahlen/bankleitzahlen.html |
| Konto - Kontonummer | Die nationale Kontonummer eines Kunden. Die Kontonummer ist in Deutschland 2 bis 10stellig. In Österreich ist die Kontonummer 2 bis 11stellig.<br><br>Innerhalb der API wird die Kontonummer immer 10stellig (Deutschland) oder 11stellig (Österreich) dargestellt. |
| BIC | Der BIC ist die internationale Adresse einer Bank. Der BIC ist 8stellig oder 11stellig.<br><br>Beispiele für gültige BICs:<br><br>DAAEDED1030<br>BFECGPGXXXX<br>BFECGPGX<br><br>An der 5. und 6. Stelle des BICs steht das Länderkennzeichen (nach ISO 3166-1), z.B. DE für Deutschland. Eine Liste der ISO-Ländercodes finden Sie unter:<br><br>http://www.mathguide.de/projekt/doku/landcode.html |
| IBAN | Die IBAN ist die internationale Kontonummer eines Kunden. Die Länge der IBAN ist von Land zu Land unterschiedlich. In Deutschland hat die IBAN 22 Stellen und in Österreich 20 Stellen.<br><br>Die ersten beiden Buchstaben stellen den ISO-Ländercode (DE für Deutschland) dar. Die Stellen 3 und 4 stellen die Prüfzahl nach ISO 7064.<br><br>Weitere Informationen erhalten Sie über den Link:<br><br>http://de.wikipedia.org/wiki/International_Bank_Account_Number |
| CI - Gläubigeridentifikation | Jeder, der mittels SEPA Lastschriften einziehen will, benötigt von der Europäischen Zentralbank eine Gläubigeridentifikation, sprich CI. In Deutschland wird die CI von der Deutschen Bundesbank vergeben.<br><br>Die CI ist von Land zu Land unterschiedlich lang. In Deutschland hat die CI eine Länge von 18 Zeichen.<br><br>Nähere Informationen zur CI und deren Aufbau in Deutschland finden Sie unter: |

| ***Begriff*** | ***Erklärung*** |
|---|---|
| | http://www.bundesbank.de/Navigation/DE/Kerngeschaeftsfelder/Unbarer_Zahlungsverkehr/SEPA/Glaeubiger_Identifikationsnummer/glaeubiger_identifikationsnummer.html |
| Land - Länderkennzeichen | Länderkennzeichen werden immer 2stellig nach ISO 3166-1 angegeben. Eine Liste der Länderkennungen nach ISO finden Sie unter:<br><br>http://www.mathguide.de/projekt/doku/landcode.html |
| SEPA Instrumente | Es gibt für den Zahlungsverkehr drei unterschiedliche SEPA-Instrumente:<br><br>CT Credit Transfer - Überweisugnen<br>DD Core Direct Debit - Basislastschrift. Die Basislastschrift entspricht der bisherigen Lastschrift mit Einzugsermächtigung.<br>B2B B2B Direct Debit - Firmenlastschrift. Die Firmenlastschrift entspricht der bisherigen Lastschrift mit Abbuchungsauftrag.<br><br>Hinweis:<br><br>DD-Lastschriften und B2B-Lastschriften dürfen innerhalb einer physikalischen SEPA XML-Datei nicht gemischt werden, obwohl dies technisch möglich wäre. Eine gemischte Datei würde vom Banksystem abgelehnt werden. |
| Unter SEPA erreichbare Banken | Die Liste der unter SEPA erreichbaren Banken finden Sie unter:<br><br>http://www.bundesbank.de/Navigation/DE/Kerngeschaeftsfelder/Unbarer_Zahlungsverkehr/SEPA/SCL_Directory/scl_directory.html |
| Spezifikation der SEPA-Datenformate | Die technische Spezifikation der SEPA-Datenformate finden Sie unter:<br><br>http://www.ebics.de/index.php?id=77 |

# 5 Testprogramm DLLDemo.exe

Das Testprogramm DLLDemo.exe dient dazu das grundsätzliche Verhalten der DLL zu testen. Das Testprogramm steht unter www.sepa-tool.de im Menüpunkt „DLL-Version“ zur Verfügung.

Mit dem Testprogramm können einige grundlegende Funktionen der DLL aufgerufen werden. Das Testprogramm selbst hat dabei keine eigene Intelligenz, sondern dient nur zur Erfassung der zu übergebenden Datenstrukturen.

Alle Rückgabewerte entsprechen den hier dokumentierten Rückgabewerten. Bitte verwenden Sie das Testprogramm nur gemeinsam mit dieser Dokumentation.

**Hinweis:**

Das Testprogramm DLLDemo.exe muss sich im gleichen Verzeichnis wie die Datei SepaTools.dll befinden.

# 6 SepaTools_Version

## *6.1 Zweck der Funktion*

Die Funktion gibt Versionsinformationen von SepaTools DLL (API) zurück. Diese Funktion kann ohne vorheriges Initialisieren der API aufgerufen werden kann.

## *6.2 Funktionsaufruf*

**void SepaTools_Version(XMLVersionStruct *VersionStruct)**

## *6.3 Parameter*

***VersionStruct*** Dies ist ein Pointer auf die Struktur XMLVersionStruct. In dieser Struktur werden die Versionsinformationen wieder zurückgegeben.

Die Struktur ist nachfolgend dargestellt.

```
struct    XMLVersionStruct

{   char    Datum[8+1]          Versionsdatum im Format TTMMJJJJ.
    int     Version             Versionsnummer
    int     SubVersion          Unter-Versionsnummer
    char    Versionsstring[20+1] Version und Unterversion als String dargestellt.
}
```

## *6.4 Returncodes*

keine.

# 7 SepaTools_SetLogFile

## *7.1 Zweck der Funktion*

Diese Funktion kann vor allen anderen Funktionen aufgerufen werden. Damit wird die Erstellung einer Log-Datei initiiert. Die Log-Datei wird in eine Textdatei geschrieben.

Innerhalb des Programmablaufes kann ein mehrfacher Aufruf dieser Funktion mit unterschiedlichen Parametern erfolgen.

## *7.2 Funktionsaufruf*

**int SepaTools_SetLogFile(const char *Path, int Action, int Inhalt, int Flush)**

## *7.3 Parameter*

***Path***

Bitte übergeben Sie hier einen Pointer auf den Pfad (Verzeichnis und Dateiname), der für die Erstellung der Logdatei verwendet werden soll.

Wenn der Dateiname ein oder mehrere Fragezeichen (**?**) enthält, so wird der übergebene Dateiname mit einer eindeutigen Nummerierung (mit Datums- und Zeitinformationen) ergänzt. Das ist hilfreich, wenn bei mehrfachen Aufrufen der Funktion SepaTools_Init die Logdatei nicht überschrieben werden soll. Die Fragezeichen werden aus dem Dateinamen entfernt.

Beispiel:

Übergebener Pfad(Dateiname):

**C:\Test\TestLogFile?.txt**

Erzeugter Dateiname (z.B.):

**C:\Test\TestLogfile_20141215_155914_938528.txt**

***Action***

Geben Sie bitte an, welche Aktion ausgeführt werden soll. Folgende Werte stehen zur Verfügung:

**1** Es wird eine neue Log-Datei mit dem angegebenen Dateinamen erzeugt. Wenn bereits eine Datei mit dem angegebenen Dateinamen existiert, so wird diese überschrieben.

**2** Eine bereits bestehende Log-Datei wird mit neuen Einträgen erweitert. Existiert die Log-Datei mit dem angegebenen Dateinamen nicht, bricht die Funktion mit einer Fehlermeldung ab.

Dann ist vorher die gleiche Funktion mit dem Parameter 1 (neue Datei) aufzurufen.

**3** Das Schreiben in die Log-Datei wird beendet. Die Datei wird dabei geschlossen. Diese Vorgehensweise ist sinnvoll, wenn innerhalb eines

Programmablaufs nur bestimmte Passagen in die Log-Datei geschrieben werden sollen.

**4** Die Log-Datei wird geschlossen. Dieser Funktionsaufruf hat die gleiche Wirkung wie der Aufruf mit dem Wert=3. Der Aufruf mit dem Wert=4 erfolgt automatisch im Rahmen der Funktion SepaTools_Free.

Der Unterschied der 3 und 4 ist lediglich informativ. Der Wert 3 signalisiert, dass die Aufzeichnung vorübergehend unterbrochen wurde, der Wert 4, dass die Datei endgültig geschlossen wurde.

***Inhalt*** Über diesen Parameter geben Sie an, in welcher Detailtiefe die Aufzeichnungen erfolgen sollen. Dabei gibt es drei Abstufungen.

**Achtung:** **Die einzelnen Abstufungen werden bitweise miteinander verknüpft!**

1 Unter Verwendung dieses Wertes werden alle Informationen protokolliert. Informationen bedeuten keine Fehler, sondern dokumentieren einzelne Datenfelder die an die entsprechende Funktion übergeben werden.

2 Dieser Wert definiert die Protokollierung von Hinweisen. Hinweise erfolgen, wenn z.B. Werte verändert wurden.

4 Unter Verwendung dieses Parameters werden Fehler protokolliert.

8 Zusätzlich zur Ausgabe in eine Textdatei werden Meldungsfenster angezeigt. Die Anzeige kann innerhalb jedes Meldungsfensters abgebrochen werden.

Die Verwendung dieses Wertes ist insbesondere sinnvoll, wenn Sie für die Funktion SepaTools_SetLogFile die übergebenen Parameter prüfen wollen, da zu diesem Zeitpunkt naturgemäß kein LogFile erstellt werden kann.

**Hinweis:**

**Wenn Sie alle möglichen Protokollierungen erhalten wollen, so übergeben Sie den Wert 15 (1 + 2 + 4 + 8).**

**Flush** Über diesen Parameter definieren Sie das Verhalten beim Schreiben eines Eintrags. Dabei gibt es folgende Varianten:

**0** Es erfolgt nach dem Schreiben des Eintrags keine weitere Aktion. Sollte das Programm nicht korrekt beendet werden können, so können Einträge verloren gehen, da die Datei erst innerhalb des Funktionsaufrufs SepaTools_Free geschlossen wird.

**1** Es erfolgt nach jedem Schreiben in die Protokolldatei ein „Flushen“ der Daten auf den Datenträger. Das Risiko, dass bei einer unerwarteten Programmunterbrechung Daten verloren gehen ist damit geringer.

**2** Die Log-Datei wird nach jedem Schreiben geschlossen und sofort wieder neu geöffnet. Damit können keine Daten verloren gehen.

Wird hier ein Parameter ungleich von 0 bis 2 übergeben, so wird dieser Parameter intern auf den Wert 2 gesetzt.

## *7.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die Angabe für Pfad und Dateinamen ist kürzer als 5 Zeichen. Geben Sie einen längeren Pfad/Dateinamen an. |
| -2 | Der Parameter Action liegt nicht im Bereich von 1 bis 4. Bitte überprüfen Sie den Parameter. |
| -3 | Der Parameter Inhalt liegt nicht im Bereich von 1 bis 15. Bitte überprüfen Sie den Parameter. |
| -4 | Das Laufwerk des übergebenen Pfades ist nicht bereit. |
| -5 | Der übergebene Pfad existiert nicht. |
| -6 | Das übergebene Verzeichnis ist schreibgeschützt. |
| -7 | Die Log-Datei konnte nicht erzeugt werden. |
| -8 | Die Log-Datei konnte nicht erweitert (Parameter Action=2) werden. |

## *7.5 Beispiel einer kleinen Log-Datei*

```
--------------------------------------------------------------------------------
29.10.2014-11:36.57 SYSTEM: SEPATOOLS-LOG-FILE: C:\Test\LogFile.txt
29.10.2014-11:36.57 SYSTEM: Modus: CREATE
--------------------------------------------------------------------------------
29.10.2014-11:36.57 INFO:   Init.Eintritt in die Funktion.
29.10.2014-11:36.57 INFO:   Init.Netz=0
29.10.2014-11:36.57 INFO:   Init.DataPath=
29.10.2014-11:36.57 INFO:   Init.UserPath=
29.10.2014-11:36.57 INFO:   Init.TempPath=
29.10.2014-11:36.57 INFO:   Init.Ermittelter TempPfad=C:\Users\Sepp\AppData\Local\Temp\
29.10.2014-11:36.57 INFO:   Init.Lizenz=
29.10.2014-11:36.57 INFO:   Init.Der Bankleitzahlenbestand hat eine Gültigkeit bis zum 08.12.2014.
29.10.2014-11:36.57 INFO:   Init.Ergebniscode=0
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.Eintritt in die Funktion.
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.BLZ=74061813
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.Konto=33626
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.Länge BLZ=8
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.BLZ_Neu=74061813
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.Konto_Neu=0000033626
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.BIC_Neu=GENODEF1PFK
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.IBAN_Neu=DE14740618130000033626
29.10.2014-11:37.11 INFO:   ConvertBLZKonto.Ergebniscode=0
29.10.2014-11:37.18 INFO:   Free.Eintritt in die Funktion.
29.10.2014-11:37.18 INFO:   Free.Funktion erfolgreich durchlaufen.
--------------------------------------------------------------------------------
29.10.2014-11:37.18 SYSTEM: SEPATOOLS-LOG-FILE: C:\Test\LogFile.txt
29.10.2014-11:37.18 SYSTEM: Modus: CLOSE
--------------------------------------------------------------------------------
```

# 8 SepaTools_SetLogFileInhalt

## *8.1 Zweck der Funktion*

In Verbindung mit der Funktion SepaTools_SetLogFile kann über diese Funktion innerhalb des Programmablaufs der Umfang der der Meldungen gesteuert werden. Für bestimmte Programmabschnitte kann damit die Ausgabe unterschiedlich gesteuert oder auch unterbunden werden.

## *8.2 Funktionsaufruf*

**int SepaTools_SetLogFileInhalt( int Inhalt)**

## *8.3 Parameter*

***Inhalt*** Über diesen Parameter geben Sie an, in welcher Detailtiefe die Aufzeichnungen erfolgen sollen. Dabei gibt es drei Abstufungen.

**Achtung:** **Die einzelnen Abstufungen werden bitweise miteinander verknüpft!**

0 Es erfolgt keine Ausgabe in die Log-Datei oder das Meldungsfenster.

1 Unter Verwendung dieses Wertes werden alle Informationen protokolliert. Informationen bedeuten keine Fehler, sondern dokumentieren einzelne Datenfelder die an die entsprechende Funktion übergeben werden.

2 Dieser Wert definiert die Protokollierung von Hinweisen. Hinweise erfolgen, wenn z.B. Werte verändert wurden.

4 Unter Verwendung dieses Parameters werden Fehler protokolliert.

8 Zusätzlich zur Ausgabe in eine Textdatei werden Meldungsfenster angezeigt. Die Anzeige kann innerhalb jedes Meldungsfensters abgebrochen werden.

**Hinweis:**

**Wenn Sie alle möglichen Protokollierungen erhalten wollen, so übergeben Sie den Wert 15 (1 + 2 + 4 + 8).**

## *8.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Der Parameter Inhalt liegt nicht im Bereich von 0 bis 15. Bitte überprüfen Sie den Parameter. |

# 9 SepaTools_Init

## *9.1 Zweck der Funktion*

Über diese Funktion werden grundlegende Initialisierungen der API vorgenommen. Sie muss als erste Funktion aufgerufen werden.

## *9.2 Funktionsaufruf*

```
int SepaTools_Init(const char *DataPath,
                   const char *TempPath,
                   const char *UserPath,
                   const char *Lizenz,
                   int Netz)
```

## *9.3 Parameter*

***DataPath*** Übergeben Sie hier einen Pointer auf das Verzeichnis, in dem sich die Datenbanken befinden. Es handelt sich um die Dateien Sepa.dat, Sepa.idx, BLZ.dat und BLZ.idx.

Diese Datenbanken werden für die korrekte Funktion der API benötigt. Wird hier kein Pfad übergeben, so werden die Datenbanken im aktuellen Verzeichnis gesucht.

Für den Fall, dass Sie keinen konkreten Pfad übergeben, beachten Sie bitte, dass sich das aktuelle Verzeichnis innerhalb des Programmablaufes nicht ändert!

Der Name des Verzeichnisses darf maximal eine Länge von 200 Zeichen haben.

***TempPath*** Übergeben Sie hier einen Pointer auf das Verzeichnis, in dem temporäre Dateien abgelegt werden sollen.

Wenn Sie hier einen Leerstring übergeben, so wird automatisch ein temporäres Verzeichnis ermittelt.

Der Name des Verzeichnisses darf maximal eine Länge von 200 Zeichen haben.

***UserPath*** Übergeben Sie bitte hier optional einen Pointer auf ein Verzeichnis, in dem User-Daten abgelegt werden können. User-Daten werden z.B. verwendet, wenn bei der Umsetzung von Kontonummer und Bankleitzahl in BIC und IBAN abweichende BICs oder IBANs verwendet werden sollen.

Wenn Sie diese Option nicht nutzen möchten, so übergeben Sie hier bitte den Wert Nil oder NULL. Damit wird diese Option nicht initialisiert.

Wenn Sie hier ein gültiges Verzeichnis übergeben, so erfolgt eine Plausibilitätsprüfung mit entsprechenden Return-Codes. Der Name des Verzeichnisses darf maximal eine Länge von 200 Zeichen haben.

| | |
|---|---|
| | Wenn Sie hier einen Leerstring übergeben, so wird als Verzeichnis mit dem Namen SepaUserData innerhalb der Profile erzeugt. Das Verzeichnis liegt dann beispielsweise unter C:\Dokumente und Einstellungen\User\Anwendungsdaten\SepaUserData. |
| ***Lizenz*** | Zur dauerhaften Nutzung der API ist eine Lizenzierung erforderlich. Im Rahmen der Lizenzierung erhalten Sie einen bis zu 10stelligen Lizenzcode. Dieser ist hier (nullterminiert) anzugeben.<br><br>Wenn Sie die API zu Testzwecken als Demoversion nutzen wollen, so übergeben Sie hier bitte einen Leerstring. Die API kann dann zu Evolutionszwecken genutzt werden. Mehrfach täglich erhalten Sie dann beim Aufruf dieser Funktion einen Hinweis auf die Demoversion.<br><br>Bei Angabe eines richtigen Lizenzcodes erschient dieser Hinweis nicht mehr. |
| ***Netz*** | Dieser Parameter hat keine Bedeutung mehr. Die Datenbanken arbeiten nun immer im Netzwerkmodus. |

## *9.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| 1 | Alles verlief erfolgreich. Allerdings sind Ihre Datenbanken (Sepa.dat und BLZ.dat) nicht mehr aktuell. Bitte prüfen Sie die Aktualität der Datenbanken über die Funktion SepaTools_Info. Aktuelle Datenbanken finden Sie unter www.sepa-tool.de.<br><br>**Der positive Returncode 1 stellt nur einen Hinweis dar, keinen Fehler. Bitte beachten Sie hierzu auch die Funktion SepaTools_Info.** |
| -1 | Das Verzeichnis für das Datenverzeichnis (DataPath) existiert nicht. |
| -2 | Das Verzeichnis (TempPath) für temporäre Daten existiert nicht. |
| -3 | Der übergebene Lizenzschlüssel ist ungültig. Zum Testen der API sollten Sie einen Leerstring übergeben. |
| -4 | Die Initialisierung der Datenbank ist fehlgeschlagen. |
| -5 | Der Pfad für das Datenverzeichnis (DataPath) ist zu lang, länger als 200 Zeichen. |
| -6 | Der Pfad für das temporäre Verzeichnis (TempPath) ist zu lang, länger als 200 Zeichen. |
| -7 | Das angegebene Laufwerk für das Datenverzeichnis (DataPath) ist nicht bereit. |
| -8 | Das angegebene Laufwerk für das temporäre Verzeichnis (TempPath) ist nicht bereit. |

| | |
|---|---|
| -9 | Die Datenbank mit den Sepa-Daten (Sepa.dat) konnte nicht geöffnet werden. Bitte überprüfen Sie den Parameter DataPath. |
| -10 | Die Datenbank mit dem Bankleitzahlenbestand der Deutschen Bundesbank (BLZ.dat) konnte nicht geöffnet werden. Bitte überprüfen Sie den Parameter DataPath. |
| -11 | Der Pfad für das Verzeichnis für die User-Daten (UserPath) ist zu lang, länger als 200 Zeichen. |
| -12 | Das angegebene Laufwerk für das das Verzeichnis für die User-Daten (UserPath) ist nicht bereit. |
| -13 | Das Verzeichnis für die User-Daten (UserPath) existiert nicht. |
| -14 | Als Verzeichnis wurde ein Leerstring übergeben. Das Verzeichnis in den Profilen wurde nicht gefunden. |
| -15 | Das Verzeichnis SepaUserData konnte nicht erzeugt werden. |
| -16 | Das Verzeichnis für die User-Daten kann nicht beschrieben werden. |
| -17 | Die Datenbank für die User-Daten konnte nicht erzeugt, bzw. geöffnet werden. |
| -18 | Die Demo-Version war nur 60 Tage lang gültig. Dieser Zeitraum ist abgelaufen. Bitte erwerben Sie zur weiteren Nutzung eine Lizenz. |
| -19 | Die Datenbanken passen nicht zur aktuellen Programmversion. Bitte besorgen Sie sich die aktuelle DLL und die aktuellen Datenbanken von www.sepa-tool.de. |

# 10 SepaTools_Free

## *10.1 Zweck der Funktion*

Über diese Funktion wird die API beendet. Alle über die Funktion SepaTools_Init augebauten Speicherbereiche werden wieder freigegeben. Es ist die letzte Funktion, die aufgerufen werden muss.

## *10.2 Funktionsaufruf*

**void SepaTools_Free()**

## *10.3 Parameter*

keine.

## *10.4 Returncodes*

keine.

# 11 SepaTools_SetOptions

## *11.1 Zweck der Funktion*

Über diese Funktion können zusätzliche, globale Optionen gesetzt werden. Wenn diese Funktion verwendet werden soll, so sollte sie gleich **nach** der Funktion SepaTools_Init aufgerufen werden.

Der Aufruf der Funktion ist optional.

## *11.2 Funktionsaufruf*

**int SepaTools_SetOptions(XMLOptionStruct *OptionStruct)**

## *11.3 Parameter*

***OptionStruct*** Dies ist ein Pointer auf die Struktur XMLOptionStruct. In dieser Struktur werden die Informationen übergeben.

Die Struktur ist nachfolgend dargestellt.

struct XMLOptionStruct

| | | | |
|---|---|---|---|
| { | int | FullNameSpace | Wert=1, wenn vollständiger Namespace. [1)] |
| | char | ConvertFile[255+1] | Pfad und Dateiname für zusätzliche Konvertierungsdatei. [2)] |
| | int | ConvertFlag | Diverse Steuerungsmöglichkeiten [3)] |
| | int | UebwSicht | Sofort fällige Überweisung [4)] |
| | int | EilUebw | Taggleiche Überweisung [5)] |
| | int | NumMsgID | Rein numerische Message-ID erzeugen [6)]. |
| | int | FullPurp | Verwendungszweck nicht verändern [7)] |
| | int | SammlerTrennen | Für jeden Sammler eine XML-Datei erstellen [8)] |
| | int | MaxXMLSatz | Max. Anzahl Datensätze in einer XML-Datei [9)] |
| | int | NotNameOrt | Name/Ort werden nicht aus dem BLZ-Verzeichnis entn. [10)] |
| | int | CompressXML | Komprimierte Darstellung der XML-Datei [11)] |
| | int | Umlaute | Wert=0, Umlaute umwandeln, Wert=1, Umlaute übergeben |
| | int | CheckSlash | Wert=0, Slash nicht prüfen, Wert=1, Slash prüfen [12)] |
| | int | ISOSonder | Wert=1 für Instant-Zahlung nach ISO2017 [13)] |
| | int | MT940KontoStart | Stelle an der die Kontonummer beginnt [14)] |
| | int | MT940KontoLen | Länge der Kontonummer [14)] |
| | int | MT940KontoMinLen | Wirkung gelten erst wenn die Kontonummer größer ist [14)] |
| | int | MT940MoreBuTag | Mehrere Buchungstage in einer Datei erlauben [15)] |
| | int | CamtVersion | Camt-Version 2009 oder 2019 [16)] |
| | int | NoBlankZweck | Keine Leerzeichen bei zusammengesetzten Verw.-Zweck [17)] |
| | int | NewUltimate | Optionale Darstellung des abweichenen Zahlungsempf. [18)] |
| | int | PmtInfo | Optionale Darstellung des PaymentInfo Blocks [19)] |
| | char | Reserve[928] | Reserve zur späteren Verwendung. |
| } | | | |

Zusätzliche Erklärungen:

1) Bei der Erstellung der XML-Datei wird normalerweise der Namespace in einer verkürzten Form, ohne Validierungsinformationen ausgegeben. Soll für die XML-Datei der vollständige Namespace verwendet werden, so ist hier der Wert 1 zu übergeben.

2) Wenn in diesem Parameter ein gültiger Pfad und ein gültiger Dateiname übergeben werden, so wird bei der Konvertierung von Bankleitzahl und Kontonummer zusätzlich eine CSV-Datei mit diesem Dateinamen geschrieben. Die Datei hat den Aufbau einer standardisierten IBAN-Rück-Datei.

   Beispiel:

```
"DE";;;;;74061813;0000070998;;"GENODEF1PFK";"DE61740618130000070998";;00
"DE";;;;;74351430;0000000224;;"BYLADEM1EGF";"DE57743514300000000224";;00
"DE";;;;;74061813;0000012345;;;;;11
```

   Betroffen sind nur die Funktionen SepaTools_WriteXML und SepaTools_WriteDTAtoXML.

   Einträge in die Datei werden nur geschrieben, wenn auch tatsächlich Konvertierungen vorgenommen wurden. Wurden BIC und IBAN direkt übergeben und verwendet, so erfolgt kein Eintrag in diese Datei.

   Die Beschreibung der möglichen Fehlercodes finden Sie in der Datei ZKASpec.pdf auf www.sepa-tool.de unter dem Menüpunkt „Sonstige Infos“.

3) Der Wert dieses Parameters dient diversen Steuerungsmöglichkeiten. Er ist daher bitweise zu belegen, damit in diesem einen Wert mehrere Steuerungsoptionen untergebracht werden können.

   Derzeit sind folgende Optionen (gemeinsame Verwendung möglich) definiert.

   1 Normalerweise wird bei der Funktion SepaTools_ConvertBLZKonto die Struktur XML-ConvertStruct nur gefüllt, wenn der Funktionsaufruf erfolgreich war. Wurde z.B. eine falsche Prüfziffer entdeckt, so blieb die Struktur leer.

     Wird mit der Funktion SepaTools_SetOptions im Feld ConvertFlag der Wert 1 übergeben, so wird die IBAN auch berechnet, wenn sie aufgrund von Fehlern nicht verwendet werden kann.

     Der eigentliche Rückgabecode bei der Funktion SepaTools_ConvertBLZKonto ändert sich dabei nicht.

   2 Wird der Wert 2 mitgeliefert (als Oder-Verbindung), so werden bei den Funktionen WriteXML und WriteXMLExt fehlerhafte BICs und IBANs nicht reklamiert und trotzdem in die XML-Datei geschrieben. Diese Option wirkt global, bis sie wieder verändert wird.

     Wenn Sie die Funktion XMLWriteExt verwenden, so können Sie alternativ zu dieser Möglichkeit auch den korrespondierenden Wert in der Struktur XMLWriteExtStruc belegen. Damit greift die Steuerungsmöglichkeit nicht global, sondern pro Datensatz.

   4 Wird der Wert 4 (als Oder-Verbindung) mitgeliefert, so liefern die Funktionen XMLWrite und XMLWriteExt zwar die gewohnten Rückgabecodes, aber der Datensatz wird nicht in die XML-Datei geschrieben.

     Mit dieser Option können Sie über die Funktionen XMLWrite und XMLWriteExt eine Datenstruktur auf Korrektheit prüfen, ohne sie aber in die XML-Datei zu schreiben.

     Diese Option wirkt global, bis sie wieder verändert wird. Wenn Sie die Funktion XMLWriteExt verwenden, so können Sie alternativ zu dieser Möglichkeit auch den korrespondie-

renden Wert in der Struktur XMLWriteExtStruc belegen. Damit greift die Steuerungsmöglichkeit nicht global, sondern pro Datensatz.

4) Wenn über SepaTools erzeugte XML-Überweisungsdateien mittels HBCI (FinTS) an die Bank übertragen werden sollen, so kann es vorkommen dass das Übertragungsprogramm keine Überweisungen mit einem Ausführungsdatum akzeptiert und nur „sofort fällige“ Überweisungen annimmt.

   Wenn dies der Fall ist, so setzen Sie bitte den Parameter UebwSicht auf den Wert 1. Das Ausführungsdatum wird dann intern automatisch auf das für diesen Fall definierte Ausführungsdatum 01.01.1999 gesetzt. Damit ist die Überweisung im Sinne von HBCI „sofort fällig“.

   Betroffen davon sind die Funktionen SepaTools_WriteXML und SepaTools_WriteDTAtoXML.

5) Unter der Voraussetzung, dass als XML-Version mindestens die Version 2.7 gewählt wurde, kann durch Setzen des Wertes „1“ in dieser Variablen erreicht werden, dass die Überweisung taggleich beim Empfänger gebucht wird.

   Das ist dann keine SEPA-Zahlung, sondern eine eilige Auslandszahlung, die von den Banken auch separat bepreist wird. Das SEPA XML-Format wird hier nur als Transportmittel verwendet. Es handelt sich hierbei um einen „Additional Optional Service“ (AOS), der unabhängig von den SEPA-Regularien definiert wurde.

   Wenn Sie dieses Feld mit dem Wert „2“ belegen, so können Instant-Überweisungen erzeugt werden. Siehe hierzu http://sepa-tools.de/instant-ueberweisung.html.

   Das Setzen dieser Variablen auf den Wert „1“ oder „2“ hat Auswirkungen auf die Funktionen SepaTools_WriteXML, SepaTools_ConvertCSVtoXML und SepaTools_WriteDTAtoXML.

6) Die Message-ID für eine XML-Datei kann alphanumerisch bis zu 35 Zeichen lang sein. Abweichend von dieser Norm, erwarten manche österreichische Electronic Banking Programme eine rein numerische Message-ID.

   Durch Belegen dieser Variablen mit dem Wert „1“ kann das erzwungen werden. Es wird dann eine bis zu 20 Zeichen lange, eindeutige rein numerische Message-ID erzeugt.

   Das ist unabhängig von u.U. durch die Anwendung vorgegebenen Kennungen.

   Das Setzen dieser Variablen auf den Wert „1“ hat Auswirkungen auf die Funktionen SepaTools_WriteXML und SepaTools_WriteDTAtoXML.

7) In der Struktur wird der Verwendungszweck in zwei Zeilen zu max. 70 Zeichen übergeben. Normalerweise werden dann von der DLL führende und folgende Leerzeichen abgeschnitten. die Länge des Verwendungszwecks wird daher optimiert.

   Wenn dieses Vorgehen verhindert werden soll, so setzen Sie diesen Parameter bitte auf den Wert 1. In diesem Fall wird dann der Verwendungszweck aus den beiden übergebenen Zeilen gebildet, ohne dass Anpassungen erfolgen.

   Auch das Leerzeichen, das beim Zusammenfügen der beiden Verwendungszweckzeilen normalerweise entsteht, wird hier unterdrückt.

8) Innerhalb einer XML-Datei können grundsätzlich beliebig viele Sammler gebildet werden. Dies passiert z.B. wenn Lastschriften mit unterschiedlichem Fälligkeitsdatum oder unterschiedlichen Sequenzen verarbeitet werden.

   Es gibt einige Bankanwendungen, die jeweils nur einen Sammer in der XML-Datei erlauben. Wenn Sie den Wert dieser Variablen auf 1 setzten, so wird für jeden Sammler eine eigene XML-Datei erstellt.

   Ausgehend von dem übergebenen Dateinamen für die XML-Datei werden dann neue Dateinamen gebildet.

9) Über diese Option können Sie die maximale Anzahl in einer XML-Datei begrenzen. Wird diese Anzahl erreicht, so wird eine neue XML-Datei gebildet. Der angegebene Wert muss zwischen 1 und 100.000 liegen. Ansonsten wird der Wert ignoriert.

   Ausgehend von dem übergebenen Dateinamen für die XML-Datei werden dann neue Dateinamen gebildet.

10) Bei den Funktionen SepaTools_GetBankInfo, SepaTools_GetSEPABank und SepaTools_GetBLZKonto werden aufgrund eines BICs die Daten der Bank gesucht. Dabei wird die entsprechende Bank in SCL-Directory der Deutschen Bundesbank gesucht.

    Im SCL-Directory wurden für deutsche Banken gegenüber dem Bankleitzahlenbestand der Deutschen Bundesbank zum Teil unterschiedliche Banknamen und Orte gemeldet. Aus diesem Grund wird bei diesen drei Funktionen geprüft, ob im Bankleitzahlenbestand abweichende Daten vorliegen.

    Das ist die Standardeinstellung, wenn die Variable NotNameOrt den Wert 0 hat. Für diese drei Funktionen kann über die Belegung dieser Variablen aber ein abweichendes Verhalten erzwungen werden. Die Variable kann dabei bitweise (Oder-Verknüpfung) mit folgenden Werten belegt werden.

    1 Für die Funktion SepaTools_GetBankInfo wird der aus dem SCL-Verzeichnis gefundene Name der Bank **nicht** mit einem u.U. abweichenden Namen aus dem BLZ-Verzeichnis überschrieben.

    2 Für die Funktion SepaTools_GetBankInfo erfolgt **kein** Überschreiben des Ortes der aus dem SCL-Verzeichnis gefundenen Bezeichnung. Ausnahme: Im SCL-Verzeichnis wurde kein Ort gefunden. In diesem Fall wird der Ort aus dem BLZ-Verzeichnis verwendet.

    4 Analoge Vorgehensweise für die Funktion SepaTools_GetSEPABank in Bezug auf den Banknamen.

    8 Analoge Vorgehensweise für die Funktion SepaTools_GetSEPABank in Bezug auf die Bezeichnung des Ortes.

    16 Analoge Vorgehensweise für die Funktion SepaTools_GetBLZKonto in Bezug auf den Banknamen.

    32 Analoge Vorgehensweise für die Funktion SepaTools_GetBLZKonto in Bezug auf die Bezeichnung des Ortes.

11 Normalerweise werden die XML-Dateien in lesbarer Form ausgegeben. Dazu werden in die eigentliche XML-Datei Leerzeichen und Zeilenschaltungen eingefügt.

Soll die XML-Datei in einer kompakten Form ausgegeben werden, so ist diese Variable mit dem Wert 1 zu belegen.

12 Ab der DK-Version 3.0 gibt es für die meisten Felder der XML-Datei Einschränkungen bzgl. der Schrägstriche. Dies gilt aber z.B. nicht für den Verwendungszweck, sondern vorwiegend für Referenzen.

Diese Felder dürfen nicht mit einem Schrägstrich beginnen oder enden. Ebenso sind in diesen Feldern Doppel-Schrägstriche (//) nicht erlaubt, einfache aber schon.

Mit dieser Option können Sie festlegen, ob die nicht erlaubten Zeichen (ab der DK-Version 3.0) ausgefiltert werden oder nicht.

Die Prüfung des erlaubten Zeichensatzes erfolgt in SepaTools an vielen Stellen. Diese Option greift dann aber für alle geprüften Datenfelder. Wenn Sie z.B. zwischen Referenzen und Verwendungszweck unterscheiden wollen, sollten Sie diese Option nicht setzen und das Thema selbst regeln.

13 Durch setzen diese Wertes (1) kann eine XML-Datei (Instant-Zahlung) nach dem Schema pain.001.001.008.xsd erzeugt werden. Damit kann zusätzlich zum Ausführungsdatum eine Ausführungszeit angegeben werden.

Nähere Informationen zu den Instant-Zahlungen finden Sie unter http://sepa-tools.de/instant-ueberweisungen.html.

14 In der MT940-Datei werden die Bankleitzahl und die Kontonummer des betroffenen Kontos angegeben. Aus diesen Informationen werden BIC und IBAN ermittelt.

In Einzelfällen kann es vorkommen, dass in diesen Feldern eine modifiziert Kontonummer angeben wird. Z.B. mit zusätzlichen führenden oder folgenden Nullen.

In diesen Fällen kann die IBAN nicht ermittelt werden, da die Prüfziffernberechnung fehlschlägt.

Für diesen Fall können Sie in diesem Feld die Stelle angeben, an der der relevante Teil der Kontonummer beginnt (z.B. 2 für die zweite Stelle).

Wenn im Feld ***MT940KontoStart*** ein Wert angeliefert wird, so haben Sie die Möglichkeit im Feld ***MT940KontoLen*** die Länge der relevanten Kontonummer anzugeben (z.B. 9, wenn der relevante Teil der Kontonummer 9 Ziffern lang ist).

Wenn der Wert im Feld ***MT940KontoStart*** größer als 0 ist und im Feld ***MT940KontoLen*** kein Wert übergeben wird, so wird der Wert intern auf 10 gesetzt.

In der Praxis sind Fälle aufgetaucht, dass in einer MT940-Datei Kontonummern des betroffenen Kontos teilweise mit zusätzlichen Informationen (z.B. Filialnummern) und teilweise ohne Zusatzinformationen dargestellt wurden.

Durch die Angabe (Feld ***M40KontoMinLen***), ab welcher Länge die obigen Einstellungen wirken sollen, macht es möglich, dass die korrekten, kürzeren Kontonummern ohne Anwendung der Sonderregeln ausgewertet werden können.

Die Sonderregeln greifen dann nur bei den längeren Kontonummern.

15) In einer MT940-Datei können sich Umsätze von unterschiedlichen Konten und auch unterschiedliche Buchungstage enthalten sein. Camt-Dateien enthalten aber per Definition immer nur ein Konto und einen Buchungstag.

Wenn sich das Konto ändert wird immer eine neue Camt-Datei erstellt. Durch die Belegung dieses Feldes mit dem Wert **1** können aber innerhalb eines Kontos Umsätze mehrerer Buchungstage in eine Datei geschrieben werden.

16) Derzeit wird in Deutschland die ISO Version von 2009 für das Camt-Format verwendet. Ab dem November 2021 wird die Deutsche Kreditwirtschaft auf das Format 2019 nach ISO 20022 umsteigen.

Wenn auch Sie Camt-Dateien nach der neuen Version erzeugen wollen, so belegen Sie bitte dieses Feld mit dem Wert **1**. Bei der Verwendung des bisherigen Camt-Formats ist dieser Eintrag nicht erforderlich.

Bei der Erstellung von Camt-Dateien für Österreich und Schweiz, bzw. Liechtenstein gelten besondere Regeln. In diesem Fall belegen Sie diesen Schlüsselbegriff bitte nicht.

Das richtige Dateiformat wird über die Funktion **SepaTools_SetVersionUndLand** ermittelt.

**Zur Information:**

Wenn Sie nur Camt-Dateien auslesen wollen, so haben Sie keinen Handlungsbedarf. Die unterschiedlichen Formate werden automatisch erkannt.

17) In der Camt-Datei wird der Verwendungszweck teilweise in mehreren Zeilen ausgegeben. Wenn mehr als drei Zeilen ausgegeben werden, so werden die einzelnen Zeilen, mit Leerzeichen getrennt, zu einer Zeile zusammengefasst (Zeile1).

Wenn Sie diese Option setzen (Wert=1), so entfallen beim zusammensetzen die Leerzeichen.

18) Optionale Darstellung (Wert=1) des abweichenden Zahlungsempfängers. Wird hier der Wert=1 übergeben, so wird der abweichende Zahlungsempfänger im Transaktionsblock übergeben. Ansonsten wird der abweichende Empfänger im Payment Information Block eingetragen.

19) Normalerweise wird die Payment Type Info (PmtTpInf) innerhalb des Payment Information Blocks dargestellt. Wenn Sie jedoch dieses Feld mit dem Wert 1 belegen, so wird die Payment Type Info im Transaktionsblock dargestellt.

### *11.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Der übergebene Pfad (incl. Dateiname) ist zu lang (>200 Zeichen). |
| -2 | Das im Pfad übergebene Laufwerk ist nicht bereit. |
| -3 | Das im Pfad übergebene Verzeichnis existiert nicht. |
| -4 | In das übergebene Verzeichnis kann nicht geschrieben werden. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 12 SepaTools_Info

## *12.1 Zweck der Funktion*

Die Funktion liefert Informationen über die Aktualität der Datenbanken Sepa.dat und BLZ.dat.

## *12.2 Funktionsaufruf*

**int SepaTools_Info(XMLInfoStruct *InfoStruct)**

## *12.3 Parameter*

***InfoStruct*** Dies ist ein Pointer auf die Struktur XMLInfoStruct. In dieser Struktur werden die Informationen wieder zurückgegeben.

Die Struktur ist nachfolgend dargestellt.

```
struct    XMLInfoStruct

{   char    EBADatum[8+1]     Datum der Aktualität der erreichbaren Banken (TTMMJJJJ).
    char    BLZVon[8+1]       BLZ-Bestand gültig ab diesem Datum im Format TTMMJJJJ.
    char    BLZBis[8+1]       BLZ-Bestand gültig bis zu diesem Datum im (TTMMJJJJ).
}
```

## *12.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die Informationen konnten aus der Datenbank nicht ausgelesen werden. Bitte überprüfen Sie den Datenpfad, den Sie über die Funktion Sepa-Tools_Init gesetzt haben |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 13 XML-Dateien erzeugen

Mit der API können XML-Dateien erzeugt werden. Dazu sind drei Funktionen erforderlich, die nachfolgend dargestellt sind. Der logische Ablauf stellt sich folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateXML | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_WriteXML | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages übergeben. |
| … | |
| - Aufruf von SepaTools_CloseXML | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |

Innerhalb dieser Funktionen finden umfangreiche Plausibilitätsprüfungen statt.

## *13.1 Automatische Bildung von Sammlern*

Wenn XML-Dateien durch das Übergeben von Datensätzen erzeugt werden, so werden u.U. automatisch Sammler (PmtInf-Blöcke) gebildet. Dies ist erforderlich, damit innerhalb eines Sammlers die Datensätze nach bestimmten Kriterien sortenrein sind.

Die Bildung der Sammler erfolgt nach folgenden Kriterien:

- Unterschiedliche Auftraggeber Namen
- Unterschiedliches Ausführungsdatum
- Unterschiedlicher Category Purpose Code
- Unterschiedlicher Struct-Code, wenn der strukturierte Verwendungszweck verwendet wird
- Unterschiedlicher Service Level (Standard ist SEPA), z.B. bei Eilüberweisungen
- Unterschiedlicher Auftraggeber BIC
- Unterschiedliche Auftraggeber IBAN

Bei Lastschriften zusätzlich:

- Unterschiedlicher Sequenz-Typ

# 14 SepaTools_CreateXML

## *14.1 Zweck der Funktion*

Mit dieser Funktion wird die Ausgabe von XML-Dateien initialisiert. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

## *14.2 Funktionsaufruf*

**int SepaTools_CreateXML(XMLCreateStruct *CreateStruct)**

## *14.3 Parameter*

***CreateStruct*** Hier wird ein Pointer auf die Struktur CreateStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für SEPA-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

```
struct    XMLCreateStruct

{   char    ExportPfad[255+1]      Laufwerk und Pfad für die Ausgabe der XML-Datei.
    int     Lastschrift            Überweisung (=0) oder Lastschrift (=1).
    char    XMLName[35+1]          Der Name der zu erzeugenden XML-Datei 1).
    char    MsgId[35+1]            Message-Id (Kennung) für die XML-Datei 2).
    char    EinreicherName[70+1]   Name des Einreichers der Datei (mind. 3 Zeichen).
}
```

Zusätzliche Erklärungen:

1) Unabhängig von der übergebenen oder nicht übergebenen Dateinamenserweiterung wird die Dateinamenserweiterung immer auf .xml gesetzt.

   Wird hier ein Leerstring oder der Wert Dummy.xml übergeben, so verwaltet die API den Dateienamen intern. Dies ist erforderlich, wenn die XML-Datei als Byte-Array im Speicher ausgegeben werden soll. Siehe hierzu Seite 84.

2) Die Message-Id soll eine möglichst eindeutige Identifizierung der XML-Datei sein. Wenn Sie hier einen Leerstring übergeben, so wird intern automatisch eine Message-Id gebildet und in der Struktur zurückgegeben.

### *14.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Es wurde ein formal unrichtiger Pfad (Länge < 2 Zeichen) für die Exportdatei übergeben. |
| -2 | Das angegebene Verzeichnis für die Exportdatei existiert nicht. |
| -3 | Der Datenträger, der im Pfadnamen der Exportdatei angegeben wurde ist schreibgeschützt. Die Datei kann nicht geschrieben werden. |
| -4 | Der Name des Einreichers der Daten wurde zu kurz (< 3 Zeichen) angegeben. |
| -5 | Die zum Ablauf der Funktion erforderliche temporäre Datei konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (TempPath). |
| -6 | Der Pfadname für die Exportdatei ist zu lange (>200 Zeichen). |
| -7 | Das angegebene Laufwerk im Pfad für die Exportdatei ist nicht bereit (möglicherweise kein Datenträger eingelegt). |
| -8 | Der Name für die XML-Datei wurde zu kurz (< 3 Zeichen) angegeben. |
| -9 | Die Funktion SepaTools_CreateXML wurde bereits aufgerufen. Vor einem weiteren Aufruf muss zuvor die Funktion SepaTools_CloseXML aufgerufen werden.<br><br>Die Funktionalitäten zur Erzeugung einer XML-Datei sind nicht „Multi-Tread“ fähig! |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 15 SepaTools_CreateXMLExt

## *15.1 Zweck der Funktion*

Mit dieser Funktion wird die Ausgabe von XML-Dateien initialisiert. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

Im Gegensatz zur Funktion SepaTools_CreateXML stellt die Datenstruktur dieser Funktion einen Reservebereich zur Verfügung. Damit sind künftige Erweiterungen möglich, ohne dass sich die grundsätzliche Struktur ändert.

Wir empfehlen künftig diese Funktion zur Erzeugung von XML-Dateien zu verwenden.

## *15.2 Funktionsaufruf*

**int SepaTools_CreateXMLExt( XMLCreateExtStruct *CreateStruct)**

## *15.3 Parameter*

***CreateStruct*** Hier wird ein Pointer auf die Struktur CreateStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für SEPA-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

```
struct   XMLCreateExtStruct

{   char    ExportPfad[255+1]       Laufwerk und Pfad für die Ausgabe der XML-Datei.
    int     Lastschrift             Überweisung (=0) oder Lastschrift (=1).
    char    XMLName[35+1]           Der Name der zu erzeugenden XML-Datei [1].
    char    MsgId[35+1]             Message-Id (Kennung) für die XML-Datei [2].
    char    EinreicherName[70+1]    Name des Einreichers der Datei (mind. 3 Zeichen).
    int     DoWhgSort               Wert 1 für zus. Sortierung der Datensätze nach Währung [3]
    int     PmtTypeInfoInTx         Wert 1 für PmtTpInf innerhalb der Transaktionsebene [4]
    char    CtctDtls_Nm_CH[35+1]    Kontaktdetails Name für GroupHeader, nur Schweiz [5]
    char    CtctDtls_Other_CH[35+1] Kontaktdetails Other Id für GroupHeader, nur Schweiz [5]
    int     ShortMsgId              Nummerische, 16stellige Ids verwenden [6]
    char    Reserve[996]            Reserve zur späteren Verwendung.
}
```

Zusätzliche Erklärungen:

1) Unabhängig von der übergebenen oder nicht übergebenen Dateinamenserweiterung wird die Dateinamenserweiterung immer auf .xml gesetzt.

   Wird hier ein Leerstring oder der Wert Dummy.xml übergeben, so verwaltet die API den Dateienamen intern. Dies ist erforderlich, wenn die XML-Datei als Byte-Array im Speicher ausgegeben werden soll. Siehe hierzu Seite 84.

2) Die Message-Id soll eine möglichst eindeutige Identifizierung der XML-Datei sein. Wenn Sie hier einen Leerstring übergeben, so wird intern automatisch eine Message-Id gebildet und in der Struktur zurückgegeben.

3) Normalerweise ist eine Sortierung der Datensätze nach der Währung nicht erforderlich, da die Währung grundsätzlich EUR ist. Werden jedoch XML-Dateien für die Schweiz erstellt, so können dort unterschiedliche Währungen vorkommen.

   Wird dieser Wert mit 1 belegt (Default=0), so erfolgt aufgrund dieser zusätzlichen Sortierung eine Gruppierung (jeweils mit Payment Information Block) der Zahlungen.

4) Grundsätzlich wird die PmtTpInf (enthält u.a. die Sequenz für Lastschriften) im Payment Information Block abgebildet. Das führt dazu, dass bei einer Änderung der Sequenz (z.B. von FRST auf RCUR) ein neuer Payment Information Block gebildet wird.

   Beim Kunden (Creditor bei Lastschriften) wird für jeden Payment Information Block eine Buchung auf dem Konto erzeugt.

   Ab der DDK-Version 3.1 darf dieser Wert auf 1 (Default=0) gesetzt werden. Damit ist eine Mischung von unterschiedlichen Sequenzen im Transaktionsbereich möglich und der Creditor erhält auf seinem Konto nur eine Buchung.

5) In der Schweiz können im GroupHeader noch zusätzliche Kontaktdetails untergebracht werden. Diese beiden Werte werden nur geschrieben, wenn das Feld CtctDtls_Nm_CH belegt ist.

   Damit das Feld CtctDtls_Other_CH geschrieben wird, müssen beide Felder mit Werten gefüllt sein.

6) Die Payment Info Id und die End to End Id sind mit Zeichenketten mit bis zu 35 Zeichen definiert und zulässig.

   Die Praxis hat gezeigt, dass es Banken gibt, die hier eigene, von der Spezifikation abweichende Restriktionen haben. Sie akzeptieren nur die ersten 16 Zeichen der Id.

   Um dem Rechnung zu tragen, kann dieses Feld mit einem **Bit verknüpftem** Wert belegt werden (Oder-Verknüpfung).

   1 Die End to End Id wird durch einen 16stelligen, nummerischen Wert dargestellt.

   2 Die Payment Info Id wird durch einen 16stelligen, nummerischen Wert dargestellt.

   Wenn bei beiden Feldern die 16stellige Version gewünscht ist, so ist entsprechend der Logik der Wert 3 zu übergeben.

### *15.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Es wurde ein formal unrichtiger Pfad (Länge < 2 Zeichen) für die Exportdatei übergeben. |
| -2 | Das angegebene Verzeichnis für die Exportdatei existiert nicht. |
| -3 | Der Datenträger, der im Pfadnamen der Exportdatei angegeben wurde ist schreibgeschützt. Die Datei kann nicht geschrieben werden. |
| -4 | Der Name des Einreichers der Daten wurde zu kurz (< 3 Zeichen) angegeben. |
| -5 | Die zum Ablauf der Funktion erforderliche temporäre Datei konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (TempPath). |
| -6 | Der Pfadname für die Exportdatei ist zu lange (>200 Zeichen). |
| -7 | Das angegebene Laufwerk im Pfad für die Exportdatei ist nicht bereit (möglicherweise kein Datenträger eingelegt). |
| -8 | Der Name für die XML-Datei wurde zu kurz (< 3 Zeichen) angegeben. |
| -9 | Die Funktion SepaTools_CreateXML wurde bereits aufgerufen. Vor einem weiteren Aufruf muss zuvor die Funktion SepaTools_CloseXML aufgerufen werden.<br><br>Die Funktionalitäten zur Erzeugung einer XML-Datei sind nicht „Multi-Tread“ fähig! |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 16 SepaTools_SetContactDetails

## *16.1 Zweck der Funktion*

Mit dieser Funktion können optionale Kontakt-Details zu Ansprechpartner gesetzt werden. Diese Funktion muss, wenn sie verwendet werden soll, vor dem ersten Auftruf von SepaTools_WriteXML (oder verleichbar) gesetzt werden.

Alle Felder der Datenstruktur sind optional!

Die hier dargestellten Möglichkeiten entsprechen den ISO-Vorgaben (z.B. pain.001.001.09) mit zum Teil gekürzten Datenfeldern.

Bitte beachten Sie, dass nicht alle Banken dies so unterstützen. Manche Banken unterstützen auch nur einen Teil dieser Felder. Von deutschen Banken werden diese Felder derzeit noch nicht akzeptiert.

Bitte sprechen Sie vor dem Einsatz dieser Funktion unbedingt mit der Bank, bei der Sie die Zahlungen einreichen wollen.

**int SepaTools_SetContactDetails(int Typ, ContactStruct *ContactStruc)**

## *16.2 Parameter*

**Bitte beachten:**

**Bei der Einreichung von AZV-Zahlungen nach ISO 20022 ist die Angabe von Kontakt Details in Deutschland (Einreicherland Deutschland) nicht erlaubt. Werden diese trotzdem angegeben, so werden diese von der API automatisch ausgefiltert.**

***Typ*** Über diesen Parameter wird angegeben, zu welchen Namen die Kontakt-Details gehören. Derzeit ist nur der Wert **1** erlaubt.

**Wert=1** Die Informationen beziehen sich auf den Einreicher der Datei.

***Contact*** Hier wird ein Pointer auf die Struktur ContactStruc übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

```
struct     ContactStruct

{    char     NamePrefix[4+1]      Optionale Anrede des Kontaks [1)]
     char     Name[70+1]           Optional der Name des Kontakts
     char     Telefon[30+1]        Optional die Telefonnummer [2)]
     char     Mobil[30+1]          Optionale Mobilfunknummer [2)]
     char     Fax[30+1]            Optionale Telefaxnummer [2)]
     char     Email[35+1]          Optionale E-Mail Adresse
     char     JobTitel[35+1        Optionaler beruflicher Titel, z.B. Direktor
     char     Abteilung[35+1]      Optionale Abteilung der Kontaktperson
     char     PrefMethode[4+1]     Bevorzugte Methode für die Kommunikation [3)]
     char     ChanelTyp_1[4+1]     Codierte Informationen zum Übertragungskanal [4)]
     char     ChanelId_1[35+1]     Inhalt/Beschreibung des Übertragungskanals [4)]
```

| | | |
|---|---|---|
| char | ChanelTyp_2[4+1] | Codierte Informationen zum Übertragungskanal [4)] |
| char | ChanelId_2[35+1] | Inhalt/Beschreibung des Übertragungskanals [4)] |
| char | ChanelTyp_3[4+1] | Codierte Informationen zum Übertragungskanal [4)] |
| char | ChanelId_3[35+1] | Inhalt/Beschreibung des Übertragungskanals [4)] |
| char | ChanelTyp_4[4+1] | Codierte Informationen zum Übertragungskanal [4)] |
| char | ChanelId_4[35+1] | Inhalt/Beschreibung des Übertragungskanals [4)] |
| char | Reserve[200] | Reserve für künftige Erweiterungen |

}

Zusätzliche Erklärungen:

1) Hier kann optional die Anrede für die Kontaktperson angegeben werden. Erlaubt sind hier nur die Werte DOCT, MADM, MIKS, MISS und MIST.

2) Für die Angabe der Telefonnummer gelten strikte Vorgaben. In der Telefonnummer dürfen nur die Zeichen 0 bis 9, Klammer auf ((), Klammer zu ()), Bindestrich (-) und das Pluszeichen (+) enthalten sein.

   Die Telefonnummer muss immer mit der Landesvorwahl mit vorangestelltem Pluszeichen beginnen. Die Landesvorwahl kann 1stellig bis 3stellig angegeben werden.

   Beispiele: +01455-65412 oder +049(089)456789821

   Telefonnummern, die der Regel nicht entsprechen führen zu einem negativen Returncode.

3) Erlaubt sind hier die Werte CELL, FAXX, LETT, MAIL, PHON

4) Hier können optional Angaben zum Übertragungskanal der Zahlung oder zur Software des Erstellerprogramms gemacht werden.

   Ab dem Jahr 2022 erwartet hier insbesondere die Schweiz (bei Einreichungen in der Schweiz) hier Angaben zur Software. Gemeinsam mit der ChanelId kann diese Kombination bis zu viermal angegeben werden.

   **Beispiel für die Schweiz:**

| *ChanelTyp* | *Inhalt von ChanelId* |
|---|---|
| NAME | Name/Bezichnung der Software |
| PRVD | Name des Software Herstellers |
| VRSN | Version der Software |
| SPSV | Version der Swiss Payment Standards Implementation Guidlines |

   **Wichtiger Hinweis für Einreichungen in der Schweiz:**

   Für den Schweizer Zahlungsverkehr gibt es bereits in der Struktur XMLCreateExtstruct fogende Datenfelder zur Übermittlung von Kontakt-Details:

   char CtctDtls_Nm_CH[35+1] Kontaktdetails Name für GroupHeader
   char CtctDtls_Other_CH[35+1] Kontaktdetails Other Id für GroupHeader

   Wenn das Feld CtctDtls_Nm_CH[35+1] belegt ist, so werden diese beiden Felder vorrangig übernommen. Die in diesem Kapitel beschriebene Struktur wird dann nicht geschrieben.

Für die Schweiz ist das die aktuelle Regelung (bis 2022).

## *16.3 Returncodes*

0 Alles verlief erfolgreich

-1 Es wurde ein ungültiger Wert für den **Typ** übergeben.

-2 Es wurde ein ungültiger Wert für das **NamePrefix** übergeben.

-3 Es wurde ein ungültiger Wert für die **PrefMethode** übergeben.

-4 Die Telefonnummer enthält ungültige Zeichen, oder entspricht nicht dem gefordertem Aufbau.

-5 Die Mobilfunknummer enthält ungültige Zeichen, oder entspricht nicht dem gefordertem Aufbau.

-6 Die Telefaxnummer enthält ungültige Zeichen, oder entspricht nicht dem gefordertem Aufbau.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 17 SepaTools_WriteXML

## *17.1 Zweck der Funktion*

Über jeden Aufruf dieser Funktion wird genau ein Datensatz für einen Zahlungsauftrag übergeben. Es werden umfangreiche Plausibilitätsprüfungen durchgeführt.

Werden kein BIC und keine IBAN übergeben, oder sind beide übergebenen Werte ungültig, so kann eine Bankleitzahl und eine Kontonummer ersatzweise dafür übergeben werden. Die API versucht dann aus diesen Daten BIC und IBAN zu ermitteln. Die ermittelten Werte werden dann auch in der Struktur WriteStruct zurückgeliefert.

Wurden spezielle BICs und IBANs mit den dazugehörigen Bankleitzahlen und Kontonummern hinterlegt, so wird diese Übersetzungstabelle in die Ermittlung von BIC und IBAN aus Bankleitzahl und Kontonummer mit einbezogen.

**Achtung:**

**Bei dieser automatischen Ermittlung können Fehler auftreten. Eine Haftung dafür wird nicht übernommen. Es wird empfohlen, immer BIC und IBAN direkt und korrekt zu übergeben.**

Es können gemischt Zahlungsverkehrsaufträge von unterschiedlichen Auftraggebern übergeben werden. Die Aufträge werden automatisch nach Auftraggebern sortiert, so dass bei mehreren Zahlungsverkehrsaufträgen des gleichen Auftraggebers intern automatisch Sammler gebildet werden.

## *17.2 Funktionsaufruf*

**int SepaTools_WriteXML( XMLWriteStruct *WriteStruct)**

## *17.3 Parameter*

***WriteStruct***

Hier wird ein Pointer auf die Struktur WriteStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für SEPA-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

**Bankfachlicher Hinweis:**

Der Auftraggeber einer Zahlung ist immer der, der die Zahlung aktiv veranlasst. Bei Überweisungen ist dies der Zahler und bei Lastschriften ist dies der Zahlungsempfänger.

Der Empfänger der Zahlung ist damit der, der die Zahlung erhält. Bei Überweisungen ist dies der Begünstigte und bei Lastschriften ist dies der Zahlungspflichtige.

struct XMLWriteStruct

| | | | |
|---|---|---|---|
| { | char | PmInfoId[10+1] | PaymentInfoId – Kennzeichnung des Auftrags [1)]. |
| | char | AusfDatum[8+1] | Ausführungsdatum der Zahlungen im Format TTMMJJJJ. |
| | char | AuftragName[70+1] | Name des Auftraggebers mit mind. 3 Zeichen. |
| | char | AuftragBIC[11+1] | BIC des Auftraggebers [6)]. |
| | char | AuftragIBAN[35+1] | IBAN des Auftraggebers. |
| | char | AuftragAbwName[70+1] | Optional einen abweichenden Namen des Auftraggebers. |
| | char | AuftragCI[35+1] | CI des Auftraggebers, nur bei Lastschriften. |
| | char | AuftragBLZ[8+1] | Optionale BLZ des Auftraggebers. |
| | char | Auftragkonto[11+1] | Optionale Kontonummer des Auftraggebers. |
| | char | EmpfName[70+1] | Name des Empfängers der Zahlung mit mind. 3 Zeichen. |
| | char | EmpfBIC[11+1] | BIC des Empfängers [6)]. |
| | char | EmpfIBAN[35+1] | IBAN des Empfängers. |
| | char | EmpfAbwName[70+1] | Optional ein abweichender Empfängername |
| | char | EmpfBLZ[8+1] | Optionale BLZ des Empfängers. |
| | char | EmpfKonto[11+1] | Optionale Kontonummer des Empfängers. |
| | char | Purpose[4+1] | Optionale Angabe eines Zwecks der Zahlung [2)]. |
| | char | Betrag[12+1] | Betrag der Zahlung als Zeichenkette in Cent. |
| | char | EndToEndId[10+1] | Kennung der Einzelzahlung [3)]. |
| | char | MandatId[35+1] | Mandatskennung (nur) für Lastschriften. |
| | char | MandatDatum[8+1] | Datum des Mandats im Format TTMMJJJJ. |
| | char | SequenceType[4+1] | Sequence der Lastschrift [4)]. |
| | int | B2B | Art der Lastschrift [5)]. |
| | char | Zweck1[70+1] | Verwendungszweck Zeile 1 |
| | char | Zweck2[70+1] | Verwendungszweck Zeile 2 |
| } | | | |

Zusätzliche Erklärungen:

1) Zur eindeutigen Kennzeichnung des Auftraggebers wird intern eine Kennung benötigt. Übergeben Sie hier ein bis zu 10stelliges Kürzel für diese Kennung. Intern wird das Kürzel dann um das Datum, die Uhrzeit und einer laufenden Nummer ergänzt.

   Wenn Sie hier einen Leerstring übergeben, so wird die komplette Kennung automatisch gebildet.

   Da aufgrund einer möglichen internen Sortierung die PmInfoId erst zu einem späteren Zeitpunkt im Rahmen des Aufrufs von SepaTools_CloseXML gebildet wird, kann sie an dieser Stelle noch nicht über die Struktur zurückgeliefert werden. Das tatsächliche Ergebnis sehen Sie erst in der erzeugten XML-Datei.

2) Die hier möglichen Werte sind aus dem Sepa Rulebook ersichtlich. Sie können dieses Feld auch frei lassen (Leerstring), da es optional ist und nicht zwingend benötigt wird.

3) Zur eindeutigen Kennzeichnung des Zahlungsauftrages wird intern eine Kennung benötigt. Übergeben Sie hier ein bis zu 10stelliges Kürzel für diese Kennung ein. Intern wird das Kürzel dann um das Datum, die Uhrzeit und einer laufenden Nummer ergänzt.

   Wenn Sie hier einen Leerstring übergeben, so wird die komplette Kennung automatisch gebildet.

   Da aufgrund einer möglichen internen Sortierung die EndToEndId erst zu einem späteren Zeitpunkt im Rahmen des Aufrufs von SepaTools_CloseXML gebildet wird, kann sie an die-

ser Stelle noch nicht über die Struktur zurückgeliefert werden. Das tatsächliche Ergebnis sehen Sie erst in der erzeugten XML-Datei.

4) Hier wird angegeben, in welchem Rhythmus die Lastschriften ausgeführt werden sollen. Gültige Werte sind hier:

| | |
|---|---|
| OOFF | Einmalige Lastschrift. |
| FRST | Erstmalige Einreichung der Lastschrift. |
| RCUR | Wiederkehrende Einreichung der Lastschrift. |
| FNAL | Letztmalige Einreichung der Lastschrift. |

Wird hier kein Wert übergeben, so wird intern automatisch der Wert RCUR gesetzt.

5) Mit der Belegung der Variablen „B2B“ wird die Art der Lastschrift gesteuert. folgende Werte sind möglich:

0 Normale CORE-Lastschrift mit den Einreichungsfristen 3 oder 6 Tage.
1 Firmenlastschrift B2B. Dies entspricht dem bisherigen Abbuchungsverfahren.
2 Lastschrift (COR1) mit verkürzten Einreichungsfristen von 2 Tagen. Die verkürzte Einreichungsfrist gilt für erstmalige und wiederkehrende (usw.) Lastschriften gleichermaßen.

Die Belegung mit dem Wert 2 darf nur erfolgen, wenn mit der Funktion SepaTools_SetVersionUndLand die Version mindestens auf 2.7 gesetzt wurde. Wurde diese Mindestversion nicht gesetzt, so wird im Falle der Anlieferung des Wertes 2 dieser Wert intern auf den Wert 0 gesetzt.

Der Lastschrifttyp COR1 muss zwischen den Kreditinstituten bilateral vereinbart worden sein. In Deutschland und in Österreich ist dies ab November 2013 der Fall.

**Hinweis:**

**Obwohl der Parameter B2B in dieser datensatzbezogenen Struktur angegeben wird, muss die Einreichung immer „sortenrein“ erfolgen. Eine Mischung der verschiedenen Werte für B2B innerhalb einer Datei darf nicht erfolgen!**

6) Unter Verwendung der XML-Version 2.7 kann auf die Angabe des BICs verzichtet werden. Die Zahlung wird dann im Modus „IBAN Only“ ausgeführt. dies gilt ab dem November 2013 für Zahlungen innerhalb Deutschlands und soll bis 2016 für den gesamten SEPA-Raum gelten.

Unabhängig davon kann trotzdem der BIC angegeben werden.

Intern wird unter Voraussetzung der Version 2.7 geprüft, ob die IBAN gültig ist. Ist dies der Fall, so wird der Datensatz auch ohne BIC erstellt.

## *17.4 Returncodes*

**Hinweis:**

Bei Returncodes >=0 (keine Fehler), werden im Rückgabewert auch zusätzliche Informationen gespeichert. **Diese Informationen sind bitweise abgelegt**.

**Die Bitbelegungen für den Wert >= 0 im Rückgabewert haben folgende Bedeutung:**

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| 1 | Die errechnete IBAN für den Auftraggeber kann verwendet werden. Sie ist aber nicht eindeutig. Eine Rückfrage beim Kunden wird empfohlen. |
| 2 | Die errechnete IBAN für den Empfänger der Zahlung kann verwendet werden. Sie ist aber nicht eindeutig. Eine Rückfrage beim Kunden wird empfohlen. |

**Ab hier sind die Fehlercodes wieder als negative Integer-Werte auszuwerten.**

| | |
|---|---|
| -1 | Das Schreiben des Datensatzes in die temporäre Datei ist gescheitert. |
| -2 | Die Funktion SepaTools_CreateXML wurde noch nicht aufgerufen. |
| -101 | Der Name des Zahlungsempfänger wurde nicht übergeben, oder ist kürzer als 3 Zeichen. |
| -102 | Der BIC des Empfängers ist formal ungültig.<br><br>Dieser Fehlercode wird auch generiert, wenn Sie BLZ und Kontonummer in BIC und IBAN umsetzen wollen (kein BIC und keine IBAN übergeben) und die Bankleitzahl ungültig ist. Damit kann auch kein BIC ermittelt werden. |
| -103 | Es wurde entweder keine IBAN für den Empfänger übergeben, oder die IBAN ist falsch. Die IBAN kann daher nicht verwendet werden. |
| -104 | Es wurde kein Betrag übergeben, oder der Betrag ist kleiner 0 (negativ). |
| -106 | Die Bank mit dem übergebenen BIC-Code des Empfängers nimmt am SEPA-Überweisungsverfahren nicht teil. |
| -107 | Die Bank mit dem übergebenen BIC-Code des Empfängers nimmt am SEPA-Lastschriftverfahren nicht teil. |
| -108 | Bei der Länderkennung, die in der IBAN für den Empfänger übergeben wurde handelt es sich um kein SEPA-Land. Die IBAN kann daher nicht verwendet werden. |
| -109 | Der BIC-Code des Empfängers wird nicht in der Liste der erreichbaren Banken geführt. Der BIC-Code kann daher nicht verwendet werden. |
| -110 | Für die beim Empfänger übergebene Bankleitzahl kann keine IBAN ermittelt werden. |
| -111 | Die Ermittlung der IBAN aus BLZ und Kontonummer des Empfängers ist nicht eindeutig. Die IBAN kann daher nicht ermittelt werden. |
| -112 | Bei der für den Empfänger übergebenen Kontonummer ist die Prüfziffer falsch. Eine Ermittlung der IBAN kann daher nicht erfolgen. |

| | |
|---|---|
| -113 | Die übergebene Bankleitzahl des Empfängers ist zur Löschung vorgemerkt und kann daher zur Ermittlung von BIC und IBAN nicht verwendet werden. |
| -114 | Die Bank mit dem übergebenen BIC-Code des Empfängers nimmt am SEPA-B2B-Lastschriftverfahren nicht teil. |
| -120 | Es wurde bereits eine Lastschrift vom Typ CORE erstellt. Abweichende Lastschrifttypen sind in einer Datei nicht erlaubt. |
| -121 | Es wurde bereits eine Lastschrift vom Typ B2B erstellt. Abweichende Lastschrifttypen sind in einer Datei nicht erlaubt. |
| -122 | Es wurde bereits eine Lastschrift vom Typ COR1 erstellt. Abweichende Lastschrifttypen sind in einer Datei nicht erlaubt. |
| -123 | Die Bank mit dem übergebenen BIC-Code des Empfängers nimmt am SEPA-COR1-Lastschriftverfahren nicht teil. |
| -124 | Die Bank des Empfängers ist für Instant-Überweisungen über SEPA nicht erreichbar. |
| -151 | Der übergebene Name des Auftraggebers hat weniger als drei Zeichen. |
| -152 | Für den Auftraggeber wurde entweder kein BIC-Code übergeben, oder der BIC-Code ist falsch.<br><br>Dieser Fehlercode wird auch generiert, wenn Sie BLZ und Kontonummer in BIC und IBAN umsetzen wollen (kein BIC und keine IBAN übergeben) und die Bankleitzahl ungültig ist. Damit kann auch kein BIC ermittelt werden. |
| -153 | Es wurde entweder keine IBAN für den Auftraggeber übergeben, oder die IBAN ist falsch. Die IBAN kann daher nicht verwendet werden. |
| -156 | Die Bank mit dem übergebenen BIC-Code des Auftraggebers nimmt am SEPA-Überweisungsverfahren nicht teil. |
| -157 | Die Bank mit dem übergebenen BIC-Code des Auftraggebers nimmt am SEPA-Lastschriftverfahren nicht teil. |
| -158 | Bei der Länderkennung, die in der IBAN für den Auftraggeber übergeben wurde handelt es sich um kein SEPA-Land. Die IBAN kann daher nicht verwendet werden. |
| -159 | Sie wollen Lastschriften erstellen. Es wurde entweder keine oder eine ungültige CI übergeben. |
| -160 | Sie haben für Lastschriften entweder kein Mandat übergeben, oder das Mandat besteht aus ungültigen Zeichen. |
| -161 | Sie haben bei Lastschriften entweder kein Mandatsdatum übergeben, oder das Mandatsdatum ist ungültig. |
| -163 | Das Ausführungsdatum ist ungültig, oder liegt in der Vergangenheit. |

| | |
|---|---|
| -164 | Der BIC-Code des Auftraggebers wird nicht in der Liste der erreichbaren Banken geführt. Der BIC-Code kann daher nicht verwendet werden. |
| -165 | Für die beim Auftraggeber übergebene Bankleitzahl kann keine IBAN ermittelt werden. |
| -166 | Die Ermittlung der IBAN aus BLZ und Kontonummer des Auftraggebers ist nicht eindeutig. Die IBAN kann daher nicht ermittelt werden. |
| -167 | Bei der für den Auftraggeber übergebenen Kontonummer ist die Prüfziffer falsch. Eine Ermittlung der IBAN kann daher nicht erfolgen. |
| -168 | Die übergebene Bankleitzahl des Auftraggebers ist zur Löschung vorgemerkt und kann daher zur Ermittlung von BIC und IBAN nicht verwendet werden. |
| -169 | Der übergebene Sequenz-Typ ist ungültig. Übergeben Sie hier im Zweifelsfall einen Leerstring. |
| -170 | Die Bank mit dem übergebenen BIC-Code des Auftraggebers nimmt am SEPA-B2B-Lastschriftverfahren nicht teil. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 18 SepaTools_WriteXMLExt

## *18.1 Zweck der Funktion*

Dies ist eine erweiterte Funktion der Funktion SepaTools_WriteXML. Damit wird ein konzeptioneller Fehler bei der Funktion SepaTools_WriteXML bereinigt und weitere zusätzliche Datenfelder eingefügt.

Die Funktion SepaTools_WriteXML bleibt unverändert und ohne Veränderung der Datenstruktur erhalten. Bei der erweiterten Datenstruktur XMLWriteExtStruct werden nur die zusätzlichen Datenfelder beschrieben. Nicht beschriebene Datenfelder, Funktionalitäten der Funktion und Rückgabecodes sind damit identisch mit der ursprünglichen Funktion SepaTools_WriteXML.

## *18.2 Funktionsaufruf*

**int SepaTools_WriteXMLExt( XMLWriteExtStruct *WriteStruct)**

## *18.3 Parameter*

***WriteStruct*** Hier wird ein Pointer auf die Struktur XMLWriteExtStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für SEPA-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

Aufgrund der Unterstützung der XML-Erzeugung für die Schweiz hat sich die Datenstruktur geändert, bzw. wurde ergänzt. Aufgrund der differenzierteren Steuerung des Zahlungsverkehrs in der Schweiz werden für die Unterstützung dieser Funktionalität mehr Datenfelder benötigt.

Wenn die neuen Felder nicht verwendet werden sollen, so müssen Sie nur die Felder ***StructZweck***, ***StructTyp*** und ***BatchBooking*** anpassen.

Die zusätzlichen Felder wurden im Reserve-Bereich untergebracht. Die gesamte Größe der Struktur hat sich nicht verändert.

Dokumentiert werden hier nur die zusätzlichen Datenfelder.

```
struct    XMLWriteExtStruct

{   char    PmInfoId[10+1]
    char    AusfDatum[8+1]
    char    AuftragName[70+1]
    char    AuftragBIC[11+1]
    char    AuftragIBAN[35+1]
    char    AuftragAbwName[70+1]
    char    AuftragCI[35+1]
    char    AuftragBLZ[8+1]
    char    Auftragkonto[11+1]
```

| | | |
|---|---|---|
| char | EmpfName[70+1] | |
| char | EmpfBIC[11+1] | |
| char | EmpfIBAN[35+1] | |
| char | EmpfAbwName[70+1] | |
| char | EmpfBLZ[8+1] | |
| char | EmpfKonto[11+1] | |
| char | Purpose[4+1] | |
| char | Betrag[12+1] | |
| char | EndToEndId[10+1] | |
| char | MandatId[35+1] | |
| char | MandatDatum[8+1] | |
| char | SequenceType[4+1] | |
| int | B2B | |
| char | Zweck1[70+1] | |
| char | Zweck2[70+1] | |
| int | NoBICIBANCheck | Wert=1, wenn keine Prüfung von BIC und IBAN [1)] |
| int | OnlyCheck | Wert=1, wenn Datens. nicht geschr. werden soll [2)] |
| char | OrgName[70+1] | ursprünglicher Name des Einreichers der Lastschrift |
| char | OrgMandat[35+1] | ursprüngliche, bisherige Mandats-Id |
| char | OrgCI[35+1] | ursprüngliche CI des Einreichers der Lastschrift |
| char | OrgIBAN[35+1] | ursprüngliche IBAN des Zahlungspflichtigen |
| int | UseSMNDA | Wert=1, wenn SMNDA Hinweis verwenden [3)] |
| int | DoStruct | Wert=1, für strukturierten Verwendungszweck [4)] |
| char | StructZweck[35+1] | Strukturierter Verwendungszweck [4)] |
| char | StructTyp[4+1] | Typ des strukturierten Verwendungszwecks [6)] |
| int | BatchBooking | BatchBooking auf Sammlerebene explizit setzen [5)] |
| char | InstrId[35+1] | Eindeutige Transaktionsreferenz [7)] |
| char | Whg[3+1] | Währung, optional [8)] |
| char | OrgBIC[11+1] | ursprünglicher BIC des Zahlungspflichtigen |
| char | EmpfLand[2+1] | Optional Land des Zahlungsempfängers [23)] |
| char | EmpfAdrLine1_Str[35+1] | Option. Adresszeile 1 des Zahl.-empfängers [23)] |
| char | EmpfAdrLine2_Ort[35+1] | Option. Adresszeile 2 des Zahl.-empfängers [23)] |
| char | AuftragLand[2+1] | Optional Land des Auftraggebers [24)] |
| char | AuftragAdrLine1_Str[35+1] | Option. Adresszeile 1 des Auftraggebers [24)] |
| char | AuftragAdrLine2_Ort[35+1] | Option. Adresszeile 2 des Auftraggebers [24)] |
| char | CategoryPurpose[4+1] | Optional Category Purpose Code 4stellig |
| char | AusfZeit[6+1] | Optionale Ausf.-Zeit für Instant-Zahlung [29)] |
| char | StructPrtry[35+1] | Optional Priorität für strukturierten Zweck |
| char | EmpfAdrHausnummer[10+1] | Optional die Hausnummer des Empfängers [23)] |
| char | EmpfAdrPLZ[15+1] | Optional die Postleitzahl des Empfängers [23)] |
| char | AuftragAdrHausnummer[10+1] | Optionale Hausnummer des Auftraggebers [24)] |
| char | AuftragAdrPLZ[15+1] | Optionale Postleitzahl des Auftraggebers [24)] |
| char | Reserve1[292] | Reserve zur späteren Verwendung |
| Int | EmpfAdrStruct | Strukturierte Adresse verwenden [23)] |
| int | AuftragAdrStruct | Strukturierte Adresse verwenden [24)] |
| int | ChBankAdrStruct | Strukturierte Adresse für Bank verwenden [19)] |
| int | ReserveInt[12] | Reserve für spätere Verwendung (Feld mit 12 Int) |
| int | ChZVArt | Zahlungsart für Einreichung in der Schweiz [9)] |
| char | ChKonto[15+1] | Kontonummer oder Postkontonummer [10)] |
| char | ChBankBC[10+1] | Bank-Clearing Code Z.-Empf/Z.-Pflichtiger (CH) [11)] |
| char | ChBankPost[15+1] | Postkonto Z.-Empf/Z.-Pflichtiger (CH) [12)] |
| char | ChBankName[35+1] | Name der Bank Z.-Empf/Z.-Pflichtiger (CH) [13)] |
| char | ChBankLand[2+1] | Land der Bank Z.-Empf/Z.-Pflichtiger (CH) [14)] |
| char | ChBankAdrLine1_Str[35+1] | Adresszeile 1 oder Anschrift der Bank (CH) [15)] |

| | | |
|---|---|---|
| char | ChBankAdrLine2_Ort[35+1] | Adresszeile 2 oder Ort der Bank (CH) [16)] |
| char | ChBankAdrPLZ[15+1] | PLZ der Adresse der Bank (CH) [17)] |
| char | ChBankHausnummer[10+1] | Haus-Nr. der Bank Z.-Empf/Z.-Pflichtiger (CH) [18)] |
| char | ChReserve[57] | Reserve zur späteren Verwendung |
| char | ChBankClearingId[5+1] | Clearing Id für Auslandsüberweisungen (CH) [20)] |
| char | ChBankIniId[35+1] | Id des Einreichers (CH) [21)] |
| char | ChLSCreditorBC[10+1] | Bank-Clearing Code des Lastschrifteinreich. (CH) [22)] |
| char | ChAccountPrty[35+1] | Zusätzliche Inform. für das Auftraggeberkto (CH) [25)] |
| char | ChLSCreditorESR[35+1] | ESR-Teilnehmernumer des LS-Einreichers (CH) [26)] |
| int | ChPmtTpInf | PaymentTpInfo in Level B (CH) [27)] |
| int | ChNoZVArtSort | Keine Gruppierung nach schweizer ZV-Art [28)] |
| char | CHQRInfo[35+1] | Zus. Info für QR-Rechnungen im strukt. Zweck |
| char | Reserve2[384] | Reserve zur späteren Verwendung. |

}

Zusätzliche Erklärungen:

1) Wenn Sie diesen Wert auf Einzelsatz-Basis auf 1 setzen, so werden Zahlungen auch dann in die XML-Datei geschrieben, wenn BIC oder IBAN von Empfänger oder Auftraggeber falsch sind.

   Alternativ können Sie das aber auch über eine globale Option im Rahmen des Funktionsaufrufes SetOptions errreichen.

2) Wenn sie diesen auf einzelsatz-Basis auf 1 setzen, so wird dieser Datensatz **nicht** in die XML-Datei geschrieben. Die formalen Prüfungen erfolgen aber trotzdem. Die Rückgabewerte sind daher die gleichen.

   Alternativ können Sie das aber auch über eine globale Option im Rahmen des Funktionsaufrufes SetOptions errreichen.

   Hinweis:

   Die folgenden Informationen sollen den Zahlungspflichtigen über Änderungen des Mandats informieren (z.B. andere CI oder neues Mandats-Id). In wie weit diese Informationen von der Bank des Zahlungspflichtigen über den Kontoauszug auch an den Zahlungspflichtigen weitergegeben werden, ist von der Bank abhängig.

3) Wird dieser Wert auf 1 gesetzt, so wird in der XML-Datei mit dem Wert SMNDA (**S**ame **M**andat with **N**ew **D**ebtor **A**gent) indiziert dass ein bisheriges Mandat mit einer neuen Bank verwendet wird. Dieser Wert darf nur gemeinsam mit der Lastschrift Sequenz FIRST verwendet werden.

   Ende 2016 gab es in der DK-Version 3.0 eine Änderung in der Vorgehensweise. Dies wird innerhalb der DLL automatisch geregelt.

   Die Erfahrung hat gezeigt, dass diese neue Regelung noch nicht von allen Ländern umgesetzt ist. Um die alte, bisherige Regelung zu erzwingen, übergeben Sie bitte in dieem Feld den Wert 2.

4) Um anstelle des unstrukturierten Verwendungszwecks den Strukturierten Verwendungszweck zu verwenden, ist die Variable DoStruct mit dem Wert 1 zu belegen. Im Feld StructZweck kann dann eine Zahlungsreferenz übergeben werden. Es erfolgt keine Prüfung auf Gültigkeit des übergebenen Wertes.

**Bitte beachten:**

**Es kann nur der strukturierte oder der unstrukturierte Verwendungszweck übergeben werden. Beide gemeinsam sind nicht erlaubt. Die Steuerung, welcher Verwendungszweck verwendet wird erfolgt über die Variable DoStruct.**

5) Standardmäßig wird in der Payment-Info der XML-Datei der Wert für Batch-Booking nicht gesetzt. Ob beim Einreicher der Zahlungen die Sammler als Sammelbuchung oder als Einzelbuchungen auf dem Kontoauszug dargestellt werden, hängt von den Voreinstellungen der Bank des Einreichers ab.

Über den Wert BatchBooking in dieser Struktur, kann der Eintrag explizit gesetzt werden. Dabei sind folgende Werte gültig:

1 BatchBooking wird auf False gesetzt. Das bedeutet Einzelbuchungen auf dem Konto des Einreichers.

2 BatchBooking wird auf True gesetzt. Das bedeutet, dass die Buchungen des Einreichers als Sammelbuchung (eine Buchung) gebucht werden.

**Bitte beachten:**

**Ob dieses Feld sich wirklich auswirkt, hängt von den Vereinbarungen mit der Bank des Einreichers der Zahlung ab.**

6) Wenn hier kein Wert übergeben wird, so wird intern automatisch der Wert SCOR eingesetzt. In Deutschland und in Österreich ist das auch der einzige erlaubte Wert.

**Ausnahme:**

**Bei der Schweizer Zahlungsart 1 (ESR-Zahlungen) darf dieses Feld nicht belegt werden.**

Bei Lastschriften in der Schweiz (Zahlungsart 102) ist hier aber der Wert IPI oder ESR i.d.R. zu übergeben.

7) Entsprechend den deutschen Belegungsrichtlinien sollte dieser Wert nur bei Einschaltung eines technischen Dienstleisters durch diesen mit der eigenen Referenz belegt werden. In der Schweiz wird die Belegung dieses Feldes aber empfohlen.

8) Eine Belegung sollte nur bei Zahlungsaufträgen, die in der Schweiz eingereicht werden belegt werden (siehe Doku für die Schweiz). Bei Nichtbelegung wird intern automatisch EUR eingesetzt.

9) Dieses Feld darf nur bei Einreichungen in der Schweiz belegt werden. Folgende Werte sind erlaubt:

0 Standard, wenn nicht Schweiz.
1 Zahlungsart 1 in der Schweiz lt. Doku.
21 Zahlungsart 2.1 in der Schweiz lt. Doku.
22 Zahlungsart 2.2 in der Schweiz lt. Doku.
3 Zahlungsart 3 in der Schweiz lt. Doku.
4 Zahlungsart 4 in der Schweiz lt. Doku.
5 Zahlungsart 5 in der Schweiz lt. Doku.

6 Zahlungsart 6 in der Schweiz lt. Doku.
101 Lastschriften für die PostFinanz in der Schweiz (DD).
102 Lastschriften für die Banken in der Schweiz (TA).
103 Lastschriften für die Postfinanz in der Schweiz (DD).

10) Hier wird bei Schweizer Zahlungsaufträgen die Kontonummer oder Postkontonummer angegeben, wenn keine IBAN benutzt werden kann oder soll.

11) Zur Adressierung des Zahlungsempfänger oder des Zahlungspflichtigen wird in der Schweiz häufig der Bank-Clearing Code angegeben.

12) Zur Adressierung des Zahlungsempfänger oder des Zahlungspflichtigen wird in der Schweiz häufig das Postkonto der Bank des Zahlungsempfänger oder des Zahlungspflichtigen angegeben.

13) Abhängig von der Zahlungsart muss für Zahlungen aus der Schweiz heraus der Name der Bank des Zahlungsempfängers/Zahlungspflichtigen angegeben werden.

14) Abhängig von der Zahlungsart muss für Zahlungen aus der Schweiz heraus die Land der Bank des Zahlungsempfängers/Zahlungspflichtigen angegeben werden. Wird dieser Wert bei einer Zahlungsart (Schweiz), bei der die Angabe erforderlich ist nicht geliefert, so wird intern automatisch CH eingesetzt.

15) Abhängig von der Zahlungsart muss für Zahlungen aus der Schweiz heraus die Adresse der Bank des Zahlungsempfängers/Zahlungspflichtigen angegeben werden. Soll die Adresse strukturiert (in der Schweiz empfohlen) angegeben werden, so ist hier die Anschrift (Straße) der Bank des Zahlungsempfängers/Zahlungspflichtigen anzugeben.

    Wenn das Feld ***ChBankAdrStruct*** nicht belegt ist, so wird dieser Wert als Adresszeile 1 verwendet.

16) Abhängig von der Zahlungsart muss für Zahlungen aus der Schweiz heraus die Adresse der Bank des Zahlungsempfängers/Zahlungspflichtigen angegeben werden. Soll die Adresse strukturiert (in der Schweiz empfohlen) angegeben werden, so ist hier der Ort der Adresse der Bank des Zahlungsempfängers/Zahlungspflichtigen anzugeben.

    Wenn das Feld ***ChBankAdrStruct*** nicht belegt ist, so wird dieser Wert als Adresszeile 2 verwendet.

17) Abhängig von der Zahlungsart muss für Zahlungen aus der Schweiz heraus die Adresse der Bank des Zahlungsempfängers/Zahlungspflichtigen angegeben werden. Soll die Adresse strukturiert (in der Schweiz empfohlen) angegeben werden, so ist hier die Postleitzahl der Adresse der Bank des Zahlungsempfängers/Zahlungspflichtigen anzugeben.

18) Abhängig von der Zahlungsart muss für Zahlungen aus der Schweiz heraus die Adresse der Bank des Zahlungsempfängers/Zahlungspflichtigen angegeben werden. Soll die Adresse strukturiert (in der Schweiz empfohlen) angegeben werden, so ist hier die Hausnummer der Adresse der Bank des Zahlungsempfängers/Zahlungspflichtigen anzugeben.

19) Belegen Sie dieses Feld mit dem Wert **1**, dann wird die strukturierte Adresse verwendet. Wird dieser Wert nicht belegt (oder mit dem Wert **0**), so wird die unstrukturierte Adresse mit Adresszeile 1 und 2 verwendet.

20) Bei der Zahlungsart 6 (Ausland, nicht SEPA) ist die Clearing-Id entsprechend den External Codesets anzugeben. Für Deutschland lautet dieses z.B. DEBLZ.

21) Innerhalb des GroupHeaders der XML-Datei kann, bzw. muss eine Id für den Einreicher der Zahlungen angegeben werden. Bei Überweisungen ist das optional. Bei Lastschriften (Zahlungsart 101 und 102) ist diese Id zwingend. Die Id ist mit dem Kreditinstitut des Lastschrifteinreichers abzustimmen.

22) Bei der Zahlungsart 102 (Lastschriften sonstige Banken) wird der Kreditor, der Zahlungsempfänger nicht durch den BIC, sondern durch den Bank-Clearing Code identifiziert. Dieser ist hier anzugeben.

23) Optional kann die Adresse des Zahlungsempfängers übergeben werden. Wenn die Adresse übergeben werden soll, so muss zwingend auch die Länderkennung übergeben werden. Es ist auch möglich nur die Länderkennung zu übergeben.

    Soll die unstrukturierte Adresse verwendet werden, so füllen Sie optional die Zeilen ***EmpfAdrLine1_Str*** und ***EmpfAdrLine2_Ort***.

    Wenn Sie die strukturierte Adresse verwenden wollen, so entspricht das Feld ***EmpfAdrLine1_Str*** der Strasse der Adresse. Das Feld ***EmpfAdrLin2_Ort*** entspricht dann dem Ort der Adresse.

    Zusätzlich können noch optional die Felder ***EmpfAdrHausnummer*** und ***EmpfAdrPLZ*** gefült werden.

    Um die strukturierte Adresse darzustellen, muss das Feld ***EmpfAdrStruct*** mit dem Wert **1** belegt sein. Ansonsten wird die unstrukturierte Adresse mit zwei Zeilen dargestellt.

    **Bitte beachten:**

    Abweichend von der ISO-Norm ist in Deutschland nur die unstrukturierte Adresse erlaubt. Die strukturierte Adresse wird als Fehler abgewiesen.

    Bei der Einreichung in anderen Ländern ist die strukturiete Adresse i.d.R. erlaubt. In der Schweiz wird diese grundsätzlich verlangt und ist ab dem Jahr 2025 verpflichtend.

24) Optional kann die Adresse des Auftraggebers übergeben werden. Wenn die Adresse übergeben werden soll, so muss zwingend auch die Länderkennung übergeben werden. Es ist auch möglich nur die Länderkennung zu übergeben.

    Soll die unstrukturierte Adresse verwendet werden, so füllen Sie optional die Zeilen ***AuftragAdrLine1_Str*** und ***AuftragAdrLine2_Ort***.

    Wenn Sie die strukturierte Adresse verwenden wollen, so entspricht das Feld ***AuftragAdrLine1_Str*** der Strasse der Adresse. Das Feld ***AuftragAdrLin2_Ort*** entspricht dann dem Ort der Adresse.

    Zusätzlich können noch optional die Felder ***AuftragAdrHausnummer*** und ***AuftragAdrPLZ*** gefült werden.

    Um die strukturierte Adresse darzustellen, muss das Feld ***AuftragAdrStruct*** mit dem Wert **1** belegt sein. Ansonsten wird die unstrukturierte Adresse mit zwei Zeilen dargestellt.

**Bitte beachten:**

Abweichend von der ISO-Norm ist in Deutschland nur die strukturierte Adresse erlaubt. Die unstrukturierte Adresse wird als Fehler abgewiesen.

Bei der Einreichung in anderen Ländern ist die strukturiete Adresse erlaubt. In der Schweiz wird diese grundsätzlich verlangt und ist ab dem Jahr 2025 verpflichtend.

25) In der Schweiz ist es möglich für das Konto des Auftraggebers zusätzliche Informationen (z.B. Art der Verbuchung, z.B. den Wert CND bei der Berner Kantonalbank) anzuliefern.

Sind in unterschiedlichen Zahlungen innerhalb einer Datei in diesem Feld Werte enthalten, die sich in den ersten 10 Zeichen unterscheiden, so erfolgt die Bildung eines neuen Payment Information Blocks.

26) Wird im Feld ***StructTyp*** der Wert ESR übergeben, so muss in diesem Feld die ESR-Teilnehmernummer des Lastschrifteinreichers angegeben werden.

27) Die Payment-Type-Informationen können sowohl in Level B und in Level C der XML-Datei geschrieben werden. Die Empfehlung lt. SIX ist für die Zahlungsarten 1, 2.1 und 2.2 die Belegung des Level C.

Einige Banken haben aber mit Ihrer Buchungslogigk hier Probleme. Wird der Wert dieses Feldes auf „1“ gesetzt, so werden diese Informationen in Level B geschrieben.

28) Wird der Wert dieses Feldes auf „1“ gesetzt, so erfolgt innerhalb einer XML-Datei keine Gruppierung nach Zahlungsarten mehr statt.

29) Unter Verwendung der ***Option ISOSonder*** innerhalb der Funktion ***SetOptions***, kann bei Instant Zahlungen neben dem Ausführungsdatum auch eine Ausführungszeit angegeben werden. Damit wird die Version pain.001.001.08 erzeugt.

Bit der DK Version 3.7 (Wert 8 bei SepaTools) ist das ebenfalls möglich. Die Option ***ISOSonder*** darf dann aber **nicht** gesetzt werden. Es wird dann eine Version pain.001.001.09 erzeugt.

## *18.4 Returncodes (zusätzlich)*

| | |
|---|---|
| -116 | Die zur Anzeige der Mandatsänderung übergebene ursprüngliche IBAN ist ungültig. |
| -117 | Die Anzeige der Kennung SMNDA darf nur in Verbindung mit dem Sequenztyp FRST erfolgen. |
| -118 | Der ursprüngliche Name des Kreditors darf nicht mit dem aktuellen Namen des Kreditors identisch sein. |
| -119 | Das ursprüngliche Mandat darf nicht mit dem aktuellen Mandat identisch sein. |
| -173 | Die zur Anzeige der Mandatsänderung übergebene ursprüngliche CI ist ungültig. |

| | |
|---|---|
| -174 | Die ursprüngliche CI darf nicht identisch mit der aktuell verwendeten CI sein. |
| -301 | Die angegebene Zahlungsart ist ungültig. Bei Überweisungen muss diese 1,21,22,3,4,5,6 sein. Bei Lastschriften 101,102 oder 103. |
| -302 | Bei der angegeben Zahlungsart 1 oder 102 muss der strukturierte Verwendungszweck verwendet werden. |

# 19 SepaTools_CloseXML

## *19.1 Zweck der Funktion*

Mit dieser Funktion wird die eigentliche XML-Datei aus der von SepaTools_WriteXML erzeugten temporären Datei erzeugt und in die definierte Datei geschrieben.

## *19.2 Funktionsaufruf*

**int SepaTools_CloseXML( XMLCloseStruct *CloseStruct)**

## *19.3 Parameter*

| | |
|---|---|
| ***CloseStruct*** | Hier wird ein Pointer auf die Struktur CloseStruct übergeben. Die Struktur ist nachfolgend dargestellt. Die Struktur wird leer übergeben und von der API mit Informationen für die Applikation gefüllt. |

```
struct    XMLCloseStruct

{   int       Anzahl                Anzahl der erzeugten Datensätze.
    char      SummeBetrag[12+1]     Gesamtsumme der Beträge in der XML-Datei als
                                    Zeichenkette in Cent.
}
```

## *19.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die XML-Datei konnte nicht erzeugt/initialisiert werden. Bitte setzen Sie sich mit dem Hersteller in Verbindung. |
| -2 | Beim Erzeugen der einzelnen Datensätze ist ein Fehler aufgetreten. Bitte setzen Sie sich mit dem Hersteller in Verbindung. |
| -3 | Beim Schreiben der XML-Datei auf den Datenträger ist ein Fehler aufgetreten. |

Die folgenden Returncodes -4 bis -7 können nur auftreten, wenn im „Speichermodus“ gearbeitet wird (es wurde kein Dateiname für die XML-Datei übergeben). Die XML-Datei wird nicht als Datei, sondern über die Funktion SepaTools_XMLGetData (siehe auf Seite 84) im Speicher übergeben.

| | |
|---|---|
| -4 | Es stehen keine Daten zur Ausgabe der XML-Datei im Speicher zur Verfügung. |
| -5 | Die Größe der XML-Daten ist größer als die maximal erlaubte Größe von derzeit 500 MB. |
| -6 | Der Speicherbereich zu Übergabe der Daten konnte nicht belegt werden. Möglicherweise steht zu wenig Hauptspeicher zur Verfügung. |
| -7 | Die Ausgabe der XML-Daten in den Speicherbreich ist fehlgeschlagen. |
| -8 | Es konnten keine Datensätze geschrieben werden, da alle Datensätze fehlerhaft waren. |
| -9 | Die temporäre Datei konnte nicht geöffnet werden. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 20 XML-Unterstützung für die Schweiz

Die Schweiz hat einen Sonderstatus innerhalb von SEPA, da die Schweiz weder in der EU ist und auch keinen Euro hat. Das hat Auswirkungen auf den Zahlungsverkehr, der bei Schweizer Banken eingereicht wird.

Die für die Schweiz erstellten XML-Dateien können neben der Schweiz auch bei Banken in Liechtenstein eingereicht werden.

Die Erweiterungen beziehen sich aktuell auf das Erstellen von XML-Dateien mit der Funktion SepaTools_WriteXMLExt.

## *20.1 Überweisungen*

Das XML-Format für die Schweizer Banken ist nicht auf reine Euro-Zahlungen wie in Deutschland oder Österreich beschränkt. Es gibt unterschiedliche Zahlungsarten, die teilweise in den historischen Zahlungsarten der Schweiz begründet sind.

Die inzwischen gewohnte Darstellung der Zahlungsverkehrspartner über BIC und IBAN ist hier nur eine von mehreren Möglichkeiten. Ebenso kann i.d.R. als Währung neben dem Euro (EUR) auch der Schweizer Franken (CHF) angegeben werden.

Um gültige XML-Dateien zu erstellen, ist es daher erforderlich der API die gewünschte Zahlungsart mitzuteilen. Die Unterscheidungen sind in den „Schweizer Implementation Guidelines“ für Kunde-an-Bank-Meldungen für Überweisungen im Zahlungsverkehr festgelegt. Dieses Dokument ist mit freundlicher Genehmigung der SIX Interbank Clearing AG, Zürich auf der Webseite www.sepa-tools.de hinterlegt.

Innerhalb einer XML-Datei können die verschiedenen Zahlungsarten für Überweisungen gemischt werden. Die API führt automatisch eine Sortierung durch, bei der auch die Zahlungsart berücksichtigt wird.

Nach der Sortierung werden entsprechende Payment Information Blöcke gebildet, so dass gleiche Zahlungsarten immer einem Block zugeordnet werden.

Die Payment-Info-Id muss sich von Block zu Block unterscheiden. Dies macht die API standardmäßig automatisch. Wenn Sie die Funktion SepaTools_SetId einsetzen, müssen Sie auf diese Thematik selbst achten.

Folgende Zahlungsarten werden dabei unterschieden:

### 20.1.1 Inlandszahlungen

| Zahlungsart | 1 | 2.1 | 2.2 | 3 | 4 |
|---|---|---|---|---|---|
| Titel | ESR | ES 1-stufig | ES 2-stufig | IBAN/Postkonto und BC/BIC | Fremdwährung |
| Bemerkung | | Postkonto des Zahlungs-empfängers | IBAN oder Bankkonto des Zahlungs-empfängers | | |
| Payment Method | TRF/TRA | TRF/TRA | TRF/TRA | TRF/TRA | TRF/TRA |
| Local Instrument | CH01 | CH02 | CH03 | Darf nicht geliefert werden | Darf nicht geliefert werden |
| Service Level | Darf nicht SEPA sein | Darf nicht SEPA sein | Darf nicht SEPA sein | Darf nicht SEPA sein | Darf nicht SEPA sein |
| Creditor Account | ESR-TNR | Postkonto | IBAN (oder Bankkonto) oder Codierzeile | IBAN oder Postkonto oder Bankkonto | IBAN oder Postkonto oder Bankkonto |
| Creditor Agent | Darf nicht geliefert werden | Darf nicht geliefert werden | V1: BC<br>V2: BC und Postkonto der Bank<br>V3: Postkonto der Bank und Name der Bank | V1: BC<br>V2: BIC Inland | V1: BIC Inland<br>V2: BC und Name und Adresse FI<br>V3: Name und Adresse Finanz-institut Inland |
| Currency | CHF/EUR | CHF/EUR | CHF/EUR | CHF/EUR | Alle ausser CHF/EUR* |

**1** Das ESR-Verfahren ist ein Einzahlungsverfahren der Schweizer Post für Firmen, die ihren Sitz in der Schweiz haben. Teilnehmer am ESR-Verfahren erhalten ESR Teilnehmernummer, die in der XML-Datei als Empfängerkontonummer anstelle der IBAN angegeben wird.

Im strukturierten Verwendungszweck ist 27stellige ESR Referenznummer anzugeben.

Die Währung kann Euro (EUR) oder Schweizer Franken (CHF) sein.

**2.1** Bei diesem Anwendungsfall wird eine Rechnung mit rotem Einzahlungsschein (ES) zugunsten eines Postkontos beglichen, die der Zahlungsempfänger dem Zahlungspflichtigen zugestellt hat. Anstelle der IBAN wird in der XML-Datei die Postkontonummer des Zahlungsempfängers angegeben.

Der Verwendungszweck ist i.d.R. unstrukturiert darzustellen.

Die Währung kann Euro (EUR) oder Schweizer Franken (CHF) sein.

**2.2** Bei diesem Anwendungsfall wird eine Rechnung mit rotem Einzahlungsschein zugunsten eines Bankkontos beglichen, die der Zahlungsempfänger dem Zahlungspflichtigen zugestellt hat.

Das Konto des Zahlungsempfängers wird entweder über die IBAN (Inland) oder das Bankkonto adressiert. Zur Adressierung der Bank des Zahlungsempfängers gibt es drei Varianten, die wahlfrei verwendet werden können:

**V1** Angabe des Bank-Clearing Codes.
**V2** Angabe des Bank-Clearing Codes und des Postkontos der Bank des Empfängers.
**V3** Angabe des Postkontos der Bank des Empfängers und Angabe des Namens der Bank des Empfängers.

Der Verwendungszweck ist i.d.R. unstrukturiert darzustellen.

Die Währung kann Euro (EUR) oder Schweizer Franken (CHF) sein.

**3** Hierbei handelt es sich um eine Inlandszahlung in der Währung EUR oder CHF. Zur Identifizierung des Zahlungsempfängers der Zahlung kann die IBAN, das Postkonto oder ein Bankkonto dienen.

Zur Adressierung der Bank des Zahlungsempfängers gibt es drei Varianten, die wahlfrei verwendet werden können:

**V1** Angabe des Bank-Clearing Codes.
**V2** Angabe des inländischen BICs der Bank des Zahlungsempfängers.

Der Verwendungszweck ist i.d.R. unstrukturiert darzustellen.

Wenn Empfängerdaten mit BIC und IBAN mit der Zahlungsart 3 übergeben werden (V2), so wird geprüft, ob BIC und IBAN zu Schweiz (CH) oder Liechtenstein (LI) gehören. Ist dies nicht der Fall, so wird die Zahlungsart intern automatisch auf den Wert 5 gesetzt.

**4** Dieser Zahlungsart dient für Überweisungen im Inland, die in einer Fremdwährung (alle außer CHF und EUR) durchgeführt werden.

Zur Identifizierung des Zahlungsempfängers der Zahlung kann die IBAN, das Postkonto oder ein Bankkonto dienen.

Zur Adressierung der Bank des Zahlungsempfängers gibt es drei Varianten, die wahlfrei verwendet werden können:

**V1** Angabe des inländischen BICs der Bank des Zahlungsempfängers.
**V2** Angabe des Bank-Clearing Codes und des Namens und der Adresse der Bank des Zahlungsempfängers.
**V3** Angabe des Namens und der Adresse der Bank des Zahlungsempfängers.

Der Verwendungszweck ist i.d.R. unstrukturiert darzustellen.

### 20.1.2 Auslandszahlungen

| **Zahlungsart** | **5** | **6** |
|---|---|---|
| Titel | Ausland SEPA | Ausland |
| Bemerkung | | |
| Payment Method | TRF/TRA | TRF/TRA |
| Local Instrument | Darf nicht geliefert werden | Darf nicht geliefert werden |
| Service Level | SEPA | Darf nicht SEPA sein |
| Creditor Account | IBAN | IBAN oder Konto |
| Creditor Agent | BIC | V1: BIC International<br>V2: Bankcode (ohne BC) und Name und Adresse Finanzinstitut<br>V3: Name und Adresse Finanzinstitut International |
| Currency | EUR | alle* |

**5** Bei dieser Zahlung handelt es sich um eine typische europäische SEPA-Zahlung. Die Währung muss EUR sein.

Der Zahlungsempfänger ist dabei über BIC und IBAN zu adressieren.

Der Verwendungszweck ist i.d.R. unstrukturiert darzustellen.

**6** Bei dieser Zahlung handelt es sich um eine Auslandszahlung, i.d.R. außerhalb des SEPA-Raumes. Es sind alle Währungen erlaubt.

Zur Identifizierung des Zahlungsempfängers wird die IBAN oder das Bankkonto des Zahlungsempfängers verwendet.

Zur Adressierung der Bank des Zahlungsempfängers gibt es drei Varianten, die wahlfrei verwendet werden können:

**V1** Angabe des internationalen BICs der Bank des Zahlungsempfängers.
**V2** Angabe des Bank-Clearing Codes und des Namens und der Adresse der Bank des Zahlungsempfängers.
**V3** Angabe des Namens und der Adresse der Bank des Zahlungsempfängers.

Der Verwendungszweck ist i.d.R. unstrukturiert darzustellen.

### *20.2 Lastschriften*

In der Schweiz gibt es zwei unterschiedliche Lastschriftverfahren, die mit XML-Dateien abgewickelt werden können. Technisch werden diese innerhalb von SepaTools mit drei fiktiven Zahlungsarten (die in der Schweiz so nicht explizit definiert sind) dargestellt.

**101** Hierbei handelt es sich um das Lastschriftverfahren der Postfinanz (DD). Es ist weitgehend identisch mit dem Verfahren, das aus Deutschland und Österreich bekannt ist.

Zahlungsempfänger und Zahlungspflichtiger werden über BIC und IBAN definiert. Als Währung sind EUR und CHF erlaubt.

Der Verwendungszweck wird unstrukturiert dargestellt.

Dieses Verfahren wird vornehmlich für Lastschriften in SEPA-Länder verwendet. Für Lastschriften innerhalb der Schweiz kann die Zahlungsart 103 verwendet werden.

**102** Dieses Verfahren wird von den sonstigen Banken in der Schweiz verwendet (TA). Es weicht deutlich von den bekannten Verfahren aus Deutschland und Österreich ab. Insbesondere werden hier keine Mandate verwendet. Auch die Angabe einer Sequenz (z.B. RCUR) ist nicht erlaubt, bzw. möglich.

Zur Identifizierung des Kontos Zahlungsempfängers ist die IBAN oder die Kontonummer des Zahlungsempfängers zu verwenden. Die Bank des Zahlungsempfängers ist über die Bank-Clearing Nummer zu definieren.

Das Gleiche gilt für die Bankverbindung des Zahlungspflichtigen.

Der Verwendungszweck ist strukturiert zu übergeben. I.d.R. handelt es sich hierbei um eine IPI-Referenznummer oder eine ESR-Referenznummer. In diesem Zusammenhang ist als Typ für den strukturierten Verwendungszweck der Wert **IPI** oder **ESR** zu übergeben.

**103** Die Schweiz hat inzwischen das Lastschriftverfahren der Banken (TA) mit dem Lastschriftverfarhen der Postfinanz (DD) harmonisiert.Über die Zahlungsart 103 können damit Inlandslastschriften für die Postfinanz (DD) erstellt werden.

Mandate sind hier nicht erforderlich. Wenn für den Debitor ein Postbankkonto verwendet werden soll, so ist anstelle der IBAN das Feld ChKonto zu verwenden.

Bitte beachten Sie, dass aufgrund der unterschiedlichen Schemata für Lastschriften diese immer sortenrein eingereicht werden müssen. Eine Mischung von Zahlungsart 101,102 und Zahlungsart 103 (PostFinanz und Banken) ist daher nicht möglich.

## *20.3 Plausibilitätsprüfungen*

Aufgrund der unterschiedlichen Möglichkeiten für die Adressierung des Zahlungsempfängers/Zahlungspflichtigen können keine bankspezifischen Plausibilitätsprüfungen durchgeführt werden. Das aufrufende Programm ist dafür verantwortlich, dass die übergebenen Werte den Formatstandards entsprechen.

## *20.4 Validierung der XML-Dateien*

Die erzeugten XML-Dateien für die Schweiz können über die XSD-Dateien, erhältlich über http://www.six-interbank-clearing.com/de/home/standardization/iso-payments/customer-bank/implementation-guidelines.html

geprüft werden. Außerdem steht das Validierungsportal https://validation.iso-payments.ch/Validation/(S(2klb1lcz01ium1dj3tkgvfqe))/KUNDEBANK/login.aspx?Proj=1&Lang=DE

zur Verfügung.

## *20.5 Implementierung*

Die zusätzlichen Funktionen zur Unterstützung der XML-Dateien für Banken in der Schweiz werden über die Funktionen SepaTools_SetVersionUndLand und SepaTools_WriteXMLExt vorgenommen. Im Folgenden sind die Änderungen, bzw. die Ergänzungen bei diesen Funktionen dargestellt.

# 21 SepaTools_XMLSetId

## *21.1 Zweck der Funktion*

Innerhalb einer XML-Datei sind entsprechend der Spezifikationen Kennungen zu setzen. Es handelt sich hier um eindeutige Kennungen zur Kennzeichnung der einzelnen Sammler (PmInfoId) und Einzelsätze (EndToEnd-Id).

Diese Kennungen werden im Normalfall von der API intern generiert. Für den Fall, dass Sie diese Kennungen individuell setzen wollen, können Sie die Funktion SepaTools_XMLSetId verwenden.

Der Aufruf der Funktion muss direkt vor dem Aufruf der Funktion SepaTools_WriteXML erfolgen. Die mit dieser Funktion gesetzten Werte, werden innerhalb der API global gespeichert und bleiben so lange erhalten, bis neue Werte gesetzt werden.

Der Ablauf stellt sich dabei folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateXML | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_XMLSetId | Setzen der Kennungen |
| - Aufruf von SepaTools_WriteXML | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages übergeben. |
| … | |
| - Aufruf von SepaTools_CloseXML | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |

## *21.2 Funktionsaufruf*

**int SepaTools_XMLSetId(const char *PmInfo, const char *EtEInfo)**

## *21.3 Parameter*

| | |
|---|---|
| ***PmInfo*** | Übergeben Sie hier bitte eine Zeiger auf die Kennung (Sammlerkennung), die Sie hier übergeben möchten. Die Kennung kann maximal 35 Zeichen lang sein. Wenn die Kennung Zeichen enthält, die innerhalb der XML-Datei nicht zugelassen sind, so werden diese automatisch ausgefiltert.<br><br>Wenn Sie in diesem Parameter einen Leerstring übergeben, so wird die Kennung wieder automatisch (siehe Kapitel 17.3 auf der Seite 62) ermittelt.<br><br>Da insbesondere Lastschriften mit unterschiedlichen Ausführungsdatums oder Sequenzen (einmalig, wiederkehrend usw.) möglich sind, ergeben sich intern unterschiedliche Sammler. Für jeden Sammler muss eine eindeutige PmInfoId angegeben werden. |

Wenn über diese Funktion eine eigene Kennung übergeben wird, so ist die Anwendung für die korrekte Sortierung der Datensätze verantwortlich. Dabei kann es zu Unstimmigkeiten mit der internen Sortierung kommen.

Es wird dringend empfohlen in diesem Parameter einen Leerstring zu übergeben. Damit werden die PmInfoId-Kennungen intern automatisch gebildet.

***EtEInfo*** Übergeben Sie hier bitte eine Zeiger auf die Kennung (Einzelsatzkennung), die Sie hier übergeben möchten. Die Kennung kann maximal 35 Zeichen lang sein. Wenn die Kennung Zeichen enthält, die innerhalb der XML-Datei nicht zugelassen sind, so werden diese automatisch ausgefiltert.

Wenn Sie in diesem Parameter einen Leerstring übergeben, so wird die Kennung wieder automatisch (siehe Kapitel 17.3 auf der Seite 62) ermittelt.

# 22 SepaTools_XMLGetData

## *22.1 Zweck der Funktion*

Unter der Voraussetzung, dass vorher eine XML-Datei erzeugt wurde (siehe Seite 54) kann der Inhalt der XML-Datei als Speicherabbild (Byte-Array) ausgelesen werden.

Der Ablauf stellt sich dabei folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateXML | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_WriteXML | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages übergeben. |
| … | |
| - Aufruf von SepaTools_CloseXML | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |
| - Aufruf von SepaTools_XMLGetData | Der Inhalt der XML-Datei wird als Byte-Array im Speicher übergeben. |

## *22.2 Funktionsaufruf*

**int SepaTools_XMLGetData(char *Data, int *Size)**

## *22.3 Parameter*

***Data*** Nach dem Aufruf der Funktion zeigt der Pointer „Data“ auf den Speicherbereich, ab dem sich der Inhalt der XML-Datei befindet. Über die Funktion SepaTools_XMLGetData wird die Adresse des Speicherbereichs in die Variable „Data“ kopiert.

Bitte beachten:

Der erforderliche Speicherbereich wird von der API belegt. Die Anwendung braucht/darf hier keinen Speicherbereich belegen.

Der Speicherbereich für die zu übergebenden Daten wird von der API verwaltet. Bei jedem Aufruf dieser Funktion wird ein zuvor belegter Speicherbereich erst mal freigegeben, bevor ein neuer Speicherbereich belegt wird.

Beim Beenden der API über SepaTools_Free wird ein u.U. noch belegter Speicherbereich endgültig freigegeben.

***Size*** In dieser Variablen wird die Größe der Daten zurückgegeben. Die Daten sind mit einer abschließenden Null (NULL) abgeschlossen. Der hier zurückgegebene Wert für die Größe des Speicherbereichs ist daher um ein Byte größer als die tatsächlichen Daten.

## *22.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Um den Inhalt der XML-Datei als Byte-Array auszugeben, ist es erforderlich dass bei SepaTools_CreateXML für den Namen der XML-Datei ein Leerstring übergeben wurde. Dies ist hier nicht geschehen.<br><br>Wenn Sie diese Funktion nutzen möchten, muss bei SepaTools_CreateXML als Dateiname für die XML-Datei ein Leerstring ausgegeben werden. |
| -2 | Der Speicherbereich für die Übergabe der Daten wurde nicht belegt. Möglicherweise stimmt die Reihenfolge der Funktionsaufrufe nicht. Siehe hierzu Seite 84. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 23 SepaTools_XMLFreeData

## *23.1 Zweck der Funktion*

Über diese Funktion kann der Speicherbereich, der von der Funktion SepaTools_XMLGetData belegt wurde explizit freigegeben werden. Wie schon an anderer Stelle erwähnt, muss diese Aufgabe von der API nicht zwingend erledigt werden, da die API den Speicher an dieser Stelle intern verwaltet.

Diese Funktion ergibt Sinn, wenn mit sehr großen Datenvolumen gearbeitet wird und Speicherplatz gespart werden soll.

## *23.2 Funktionsaufruf*

**int SepaTools_XMLFreeData()**

## *23.3 Parameter*

keine.

## *23.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Es stand kein Speicherbereich zur Verfügung, der freigegeben werden hätte müssen. Diesen Rückgabewert erhalten Sie auch, wenn Sie diese Funktion mehrfach aufrufen, ohne dass zwischenzeitlich wieder Speicher belegt wurde. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 24 DTA-Dateien in XML-Dateien umsetzen

Mit der API können DTA-Dateien in XML-Dateien umgesetzt werden. Dabei werden auch Multi-DTA Dateien unterstützt. Unter Multi-DTA Dateien versteht man DTA-Dateien, die in einer physikalischen Datei mehrere Logische DTA-Dateien (auch gemischt zwischen Überweisungen und Lastschriften) enthalten

Auch wenn in dieser Dokumentation immer wieder vom Unterschied logischer Datei und physikalischer Datei gesprochen wird, so wird in der Praxis in den allermeisten Fällen in der physikalischen Datei nur eine logische Datei enthalten sein. Die logische Datei hat in diesen Fällen dann immer den Wert 1.

Der grundsätzliche Ablauf stellt sich dabei folgenermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_ReadDTA | Die physikalische DTA-Datei wird in eine temporäre Datei eingelesen und ggfls. intern in mehrere logische DTA-Dateien aufgeteilt. |
| - Aufruf von SepaTools_GetDTAFehler | Dieser Funktionsaufruf ist **optional**. Mit dieser Funktion können Detailinformationen ausgelesen werden, wenn schon beim Einlesen der DTA-Datei inhaltliche Fehler (z.B. Betrag=0) festgestellt wurden. |
| - Aufruf von SepaTools_GetDTAInfo | Diese Funktion muss u.U. mehrfach aufgerufen werden. Sie liefert eine Struktur mit Informationen aus der logischen DTA-Datei.<br><br>Mehrere Felder der Struktur sind erst mal nicht gefüllt und sind in der nächsten Funktion zu verwenden. |
| - Aufruf von SepaTools_PutDTAInfo | Diese Funktion muss u.U. ebenfalls mehrfach aufgerufen werden (für jede logische Datei). Übergeben wird die Struktur aus vorhergehenden Aufrufen der Funktion SepaTools_GetDTAInfo.<br><br>Der Inhalt der Struktur kann/muss dann ergänzt oder modifiziert werden. Näheres bei der Funktionsbeschreibung auf Seite 95. |
| - Aufruf von SepaTools_WriteDTAtoXML | Diese Funktion erzeugt aus der temporären Datei die eigentliche XML-Datei.<br><br>Wenn sich in der physikalischen DTA-Datei Überweisungen und Lastschriften (mehrere logische Dateien) befunden haben, so ist dieser Aufruf (und der optional folgende Aufruf) zweimal erforderlich. |
| - Aufruf von SepaTools_XMLGetData | Dieser Aufruf ist optional und wird nur verwendet, wenn die XML-Datei direkt im Speicher übergeben werden soll. |

- Aufruf von SepaTools_GetDTAProtokoll — Über den wiederholten Aufruf dieser Funktion können Protokolldaten der erfolgreich und nicht erfolgreich geschriebenen Datensätze abgeholt werden.

- Aufruf von SepaTools_FreeDTAVars — Diese Funktion gibt den intern für die Verarbeitung belegten Speicher wieder frei und löscht die temporären Dateien.

  Das ist der letzte Funktionsaufruf, der hier erforderlich ist.

Wird aufgrund der Funktion SepaTools_GetDTAInfo festgestellt, dass sich in der DTA-Datei sowohl Überweisungen als auch Lastschriften befunden haben, so ist der Block SepaTools_WriteDTAtoXML und ggfls. SepaTools_GetDTAProtokoll zweimal (einmal für Überweisungen und einmal für Lastschriften) aufzurufen.

# 25 SepaTools_ReadDTA

## *25.1 Zweck der Funktion*

Diese Funktion dient dem Einlesen einer DTA-Datei. Ebenso werden die intern erforderlichen Initialisierungen vorgenommen.

Beim Einlesen der DTA-Datei werden umfangreiche formale und inhaltliche Prüfungen vorgenommen. Bei einem ersten **nicht behebbaren** auftretenden Fehler bricht die Funktion ab (es sind hier keine automatischen Fehlerkorrekturen möglich).

Fehler, die nur einzelne Datensätze betreffen, werden von der weiteren Verarbeitung ausgeschlossen. Die fehlerfreien Datensätze werden aber weiterverarbeitet. Informationen zu diesen Datensätzen können über die Funktion SepaTools_GetDTAFehler abgerufen werden.

Optional können Sie die Struktur „DTAFehler“ auswerten. Diese liefert bei inhaltlichen Fehlern, die eine weitere Verarbeitung verhindern, Detailinformationen über den aufgetretenen Fehler.

## *25.2 Funktionsaufruf*

**int SepaTools_ReadDTA(const char *Path, DTAFehlerStruct *DTAFehler)**

## *25.3 Parameter*

***Path*** Übergeben Sie hier einen Pointer auf den vollen Dateinamen (incl. Pfad) der einzulesenden DTA-Datei (z.B. C:\Test\DTAUS0.txt).

***DTAFehlerStruct*** Hier wird ein Pointer auf die Struktur DTAFehlerStruct übergeben. Die Struktur ist nachfolgend dargestellt.

Falls beim Einlesen der DTA-Datei inhaltliche Fehler festgestellt werden und die Funktion mit einem negativen Rückgabewert abbricht, wird die Struktur mit Detailinformationen gefüllt.

Wenn Sie diese Informationen nicht auswerten wollen, so können Sie anstelle der Struktur auch einen NULL-Pointer (NULL oder Nil) übergeben.

struct DTAFehlerStruct

```
{   char    ErrorMsg[80+1]      Die Fehlermeldung im Klartext.
    char    ErrorField[10+1]    Der Name des betroffenen Feldes in der DTA-Datei 1).
    int     ErrorRec            Die Satznummer innerhalb der logischen Datei 2).
    int     NumLogDat           Die Nummer der logischen Datei innerhalb der Datei.
}
```

Zusätzliche Erklärungen:

1) Lt. DFÜ-Abkommen sind die Felder der DTA-Datei mit einem Buchstaben (A,C,E) und einer folgenden Zahl durchnummeriert (z.B. C7).

2) Die Satznummer wird pro E-Satz, C-Satz und A-Satz hochgezählt.

## *25.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Die angegebene DTA-Datei konnte nicht geöffnet werden. |
| -2 | Die angegebene DTA-Datei wurde nicht gefunden. Bitte überprüfen Sie Pfad und Dateinamen. |
| -3 | Bei der geöffneten Datei handelt es sich um keine DTA-Datei. |
| -4 | Der angegebene temporäre Pfad existiert nicht (siehe SepaTools_Init). |
| -5 | Die temporäre DTA-Datei konnte nicht erzeugt werden. |
| -6 | Fehler beim Schreiben der temporären Datei. |
| -7 | Die temporäre Datei konnte nicht geöffnet werden. |
| -8 | Fehler beim Lesen der temporären Datei. |
| -9 | Die DTA-Datei konnte nicht erfolgreich geschlossen werden. |
| -11 | Nicht behebbarer Fehler im ASatz (siehe DTAFehlerStruct). |
| -12 | Nicht behebbarer Fehler im CSatz (siehe DTAFehlerStruct). |
| -13 | Nicht behebbarer Fehler im ESatz (siehe DTAFehlerStruct). |
| -14 | ESatz gefunden, obwohl kein vorhergehender ASatz gefunden wurde. |
| -15 | ESatz gefunden, obwohl kein vorhergehender CSatz gefunden wurde. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 26 SepaTools_GetDTAFehler

## *26.1 Zweck der Funktion*

Werden beim Einlesen der DTA-Datei Fehler entdeckt, die nicht korrekt weiterverarbeitet werden können, so werden diese von der weiteren Verarbeitung ausgeschlossen. Solche Fehler wären z.B.:

- Betrag ist Null
- Bankleitzahl ist kleiner als 10000000
- Kontonummer ist kleiner als 1
- Name fehlt
- usw.

Der Aufruf dieser Funktion ist optional!

Wenn diese Funktion verwendet wird, so muss sie u.U. mehrfach aufgerufen werden.

## *26.2 Funktionsaufruf*

**int SepaTools_GetDTAFehler(DTADetailStruct *DTADetail, int *Index)**

## *26.3 Parameter*

***DTADetail*** Übergeben Sie hier einen Pointer auf die Struktur „DTADetailStruct“. Wenn beim Einlesen der DTA-Datei Fehler gefunden wurden, die eine weitere Verarbeitung der fehlerfreien Datensätze erlauben, so werden über diese Funktion Informationen zu den Fehlern ausgegeben.

***Index*** Übergeben Sie bitte hier beim ersten Aufruf der Funktion den Wert 1. Das bedeutet, dass der erste Fehlereintrag aus der Liste der erkannten Fehler in der Struktur zurückgeliefert wird.

Im Rahmen des Funktionsaufrufs wird die Variable „Index“ automatisch um den Wert 1 hochgezählt. Damit wird beim drauffolgenden Funktionsaufruf der nächste Eintrag aus der Fehlerliste zurückgeliefert.

Wird als „Index“ ein Wert kleiner 1 oder ein Wert der größer ist als die maximalen Einträge der Fehlerliste übergeben, so gibt die Funktion einen negative Rückgabewert zurück (siehe unten) und in der Variablen „Index“ die Anzahl der tatsächlich vorhandenen Einträge.

Durch gezieltes Setzen der Variablen „Index“ kann auch auf die Einträge in der Fehlerliste zugegriffen werden.

```
struct    DTADetailStruct

{   int       LogDat          Nummer der logischen Datei
    int       Rec             Satznummer innerhalb der logischen Datei
    char      Feld[5+1]       Feldnummer lt. DTA-Spezifikation 1)
    char      Fehler[50+1]    Fehlerbeschreibung im Klartext
    char      Name[27+1]      Name des Zahlungsempfängers wenn vorhanden 2)
    int       Lastschrift     Wert 1 bei Lastschriften, ansonsten 0
```

| | | |
|---|---|---|
| char | Betrag[12+1] | Betrag der Zahlung [3)] |
| char | Reserve[50+1] | Frei für spätere Erweiterungen |

}

Zusätzliche Erklärungen:

1) Lt. DTA-Spezifikation werden die Feldbezeichnungen entsprechend A-Satz, C-Satz und E-Satz durchnummeriert (z.B. C4 oder E5).

2) Wenn das Fehlen des Empfängernamens als Fehler erkannt wurde, so wird in diesem Feld auch kein Name zurückgegeben.

3) Wenn das Fehlen des Betrages als Fehler erkannt wurde, so wird in diesem Feld auch kein Betrag zurückgegeben.

## *26.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Es stehen weitere Daten zur Verfügung. Ein wiederholter Aufruf der Funktion (ohne Veränderung der Variablen „Index“) ist erforderlich. |
| -1 | Der Speicherbereich für die Fehlerliste wurde nicht reserviert. Wahrscheinlich wurde die Funktion SepaTools_ReadDTA noch nicht aufgerufen. |
| -2 | Es wurden keine Einträge in die Fehlerliste geschrieben. D.h. es wurden keine Fehler festgestellt. Dies sollte der Normalfall sein. |
| -3 | Als „Index“ wurde ein Wert kleiner als 1 übergeben. In der Variablen „Index“ wird die Anzahl der Listeneinträge in der Fehlerliste zurückgegeben. |
| -4 | Als „Index“ wurde ein Wert übergeben, der größer ist als die Anzahl der vorhandenen Listeneinträge in der Fehlerliste. In der Variablen „Index“ wird die Anzahl der Listeneinträge in der Fehlerliste zurückgegeben. |
| -5 | Der Listeneintrag ist leer. Dieser Fehlercode dürfte nicht vorkommen und deutet auf einen Programmfehler hin. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 27 SepaTools_GetDTAInfo

## *27.1 Zweck der Funktion*

Nachdem eine physikalische DTA-Datei in eine temporäre Datei eingelesen wurde, können, bzw. müssen mit dieser Funktion Informationen über die logischen Dateien ausgelesen werden. Die Funktion wird daher so oft aufgerufen, bis der Rückgabewert<>0 ist.

Die so gewonnenen Informationen dienen als Basis für die nachfolgenden Aufrufe von SepaTools_PutDTAInfo.

## *27.2 Funktionsaufruf*

**int SepaTools_GetDTAInfo(DTAInfoStruct *DTAInfo, int *First)**

## *27.3 Parameter*

***DTAInfo*** Übergeben Sie hier einen Pointer auf die Struktur „DTAInfoStruct“. Die Struktur wir mit diesem Funktionsaufruf teilweise gefüllt. Die Struktur wird auf Grund des wiederholten Funktionsaufrufes für jede logische Datei übergeben. Die Struktur ist im Nachgang dargestellt:

***First*** Beim ersten Aufruf der Funktion ist diese Variable mit dem Wert 1 zu füllen. Die Variable wird im Rahmen des ersten Aufrufs der Funktion automatisch mit 0 belegt.

struct DTAInfoStruct

| | | | |
|---|---|---|---|
| { | int | LogFileOK | Dieser Wert wird auf 1 gesetzt.[1)] |
| | int | Auftragsart | Auftragsart=0 Überweisungen, =1 Lastschriften |
| | int | LogDat | Nummer der logischen Datei.[2)] |
| | char | BLZAuftraggeber[8+1] | Die Bankleitzahl des Auftraggebers |
| | char | KontoAuftraggeber[10+1] | Die Kontonummer des Auftraggebers |
| | char | Auftraggeber[70+1] | Der Name des Auftraggebers.[3)] |
| | char | ErstDatum[8+1] | Erstellungsdatum der DTA-Datei (TTMMJJJJ). |
| | char | AusfDatum[8+1] | Ausführungsdatum, wenn vorhanden (TTMMJJJJ). |
| | int | NumSatz | Anzahl Datensätze in der logischen Datei. |
| | char | SummeBetrag[12+1] | Summe der Beträge in der logischen Datei.[4)] |
| | char | SummeKonto[17+1] | Summe der Kontonummern in der logischen Datei. |
| | char | SummeBLZ[17+1] | Summe der Bankleitzahlen in der logischen Datei. |
| | char | CI[35+1] | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | char | Mandat[35+1] | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | char | MandatDat[8+1] | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | char | Sequence[4+1] | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | int | AutoMandat | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | int | MandatStartNr | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | char | PmInfo[10+1] | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | char | EToEInfo[10+1] | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | int | B2B | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | char | Purpose[4+1] | Nicht belegt, siehe Kapitel 28.1 auf Seite 95. |
| | char | AusfZeit[8+1] | Optionale Ausführungszeit für Instant-Zahungen |
| | char | Reserve[32] | |

Zusätzliche Erklärungen:

1) Dieser Wert wird beim Auslesen der physikalischen Datei für jede logische Datei automatisch auf 1 gesetzt. Wenn Sie diese logische Datei nicht in die Konvertierung einbeziehen wollen, so müssen Sie diesen Wert vor dem Aufruf von SepaTools_PutDTAInfo auf 0 setzen.

2) Es handelt sich hier um die Nummer der logischen Datei innerhalb der eingelesenen physikalischen Datei.

   **Achtung:**

   **Dieser Wert darf nicht verändert werden, da mit diesem Wert der Zugriff auf interne Daten gesteuert wird.**

3) Obwohl im DTA-Satz der Name des Auftraggebers eine Länge von max. 27 Zeichen hat, stehen in der Struktur hierfür 70 Zeichen zur Verfügung. Sie können daher vor dem Aufruf der Funktion SepaTools_PutDTAInfo den Namen des Auftraggebers der Zahlung verändern und haben dazu 70 Zeichen zur Verfügung.

4) Beträge werden immer als Zeichenkette in Cent dargestellt. Es gibt kein Dezimalkomma oder Tausender-Separatoren. Beispiel: 1.342,76 € wird als 134276 dargestellt.

## *27.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Es stehen weitere Daten zur Verfügung. Ein wiederholter Aufruf der Funktion (mit Parameter First=0) ist erforderlich. |
| -1 | Es stehen keine Informationen über logische Dateien einer DTA-Datei zur Verfügung. Wahrscheinlich wurde die Funktion SepaTools_ReadDTA noch nicht aufgerufen. |
| -2 | In der physikalischen DTA-Datei waren keine logischen DTA-Dateien enthalten. Bitte prüfen Sie die DTA-Datei. |
| -3 | Sie sind am Ende der Daten angelangt. Es stehen keine weiteren Daten über logische DTA-Dateien mehr zur Verfügung. |
| -4 | Es ist ein logischer Speicherfehler aufgetreten (sollte nie vorkommen). Bitte informieren Sie den Hersteller. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 28 SepaTools_PutDTAInfo

## *28.1 Zweck der Funktion*

Nachdem über die Funktion SepaTools_GetDTAInfo die Informationen der logischen Dateien ausgelesen wurden, können/müssen diese u.U. modifiziert werden und wieder mit dieser Funktion zurückgeschrieben werden.

Insbesondere, wenn Sie Lastschriften konvertieren wollen, müssen die lastschriftrelevanten Felder gefüllt werden.

## *28.2 Funktionsaufruf*

**int SepaTools_PutDTAInfo(DTAInfoStruct *DTAInfo)**

## *28.3 Parameter*

***DTAInfo*** Übergeben Sie hier einen Pointer auf die Struktur „DTAInfoStruct“. Die Struktur haben Sie aufgrund der Aufrufe von SepaTools_GetDTAInfo erhalten. Die Struktur steht damit wird auf Grund des wiederholten Funktionsaufrufes für jede logische Datei übergeben.

Die Struktur ist im Nachgang dargestellt:

struct DTAInfoStruct

| | | | |
|---|---|---|---|
| { | int | LogFileOK | Dieser Wert wird auf 1 gesetzt.[1)] |
| | int | Auftragsart | Auftragsart=0 Überweisungen, =1 Lastschriften[2)] |
| | int | LogDat | Nummer der logischen Datei.[3)] |
| | char | BLZAuftraggeber[8+1] | Die Bankleitzahl des Auftraggebers |
| | char | KontoAuftraggeber[10+1] | Die Kontonummer des Auftraggebers |
| | char | Auftraggeber[70+1] | Der Name des Auftraggebers.[4)] |
| | char | ErstDatum[8+1] | Erstellungsdatum der DTA-Datei (TTMMJJJJ). |
| | char | AusfDatum[8+1] | Ausführungsdatum im Format TTMMJJJJ.[11)] |
| | int | NumSatz | Anzahl Datensätze in der logischen Datei. |
| | char | SummeBetrag[12+1] | Summe der Beträge in der logischen Datei.[5)] |
| | char | SummeKonto[17+1] | Summe der Kontonummern in der logischen Datei. |
| | char | SummeBLZ[17+1] | Summe der Bankleitzahlen in der logischen Datei. |
| | char | CI[35+1] | CI (Gläubigeridentifikation) für Lastschriften.[6)] |
| | char | Mandat[35+1] | Mandatskürzel für Lastschriften.[7)] |
| | char | MandatDat[8+1] | Datum des Mandats[7)] |
| | char | Sequence[4+1] | Häufigkeit der Ausführung bei Lastschriften[7)] |
| | int | AutoMandat | Automatische Mandatsnummerierung[8)] |
| | int | MandatStartNr | Startnummer für automatische Nummerierung[8)] |
| | char | PmInfo[10+1] | Kürzel für die Sammlerinformation[9)] |
| | char | EToEInfo[10+1] | Kürzel für die Einzelsatzinformation[10)] |
| | int | B2B | Art der Lastschrift [12)]. |
| | char | Purpose[4+1] | Purpose-Code |
| | char | Reserve[41] | Reserve für spätere Erweiterungen |
| } | | | |

Zusätzliche Erklärungen:

1) Dieser Wert wird beim Auslesen der physikalischen Datei für jede logische Datei automatisch auf 1 gesetzt. Wenn Sie diese logische Datei nicht in die Konvertierung einbeziehen wollen, so müssen Sie diesen Wert vor dem Aufruf von SepaTools_PutDTAInfo auf 0 setzen.

2) Obwohl das in der Regel nicht sinnvoll ist, können Sie durch Änderung dieses Kennzeichens die Art der Zahlung verändern. Damit können aus Überweisungen Lastschriften und umgekehrt gemacht werden.

3) Es handelt sich hier um die Nummer der logischen Datei innerhalb der eingelesenen physikalischen Datei.

   **Achtung:**

   **Dieser Wert darf nicht verändert werden, da mit diesem Wert der Zugriff auf interne Daten gesteuert wird.**

4) Obwohl im DTA-Satz der Name des Auftraggebers eine Länge von max. 27 Zeichen hat, stehen in der Struktur hierfür 70 Zeichen zur Verfügung. Sie können daher vor dem Aufruf der Funktion SepaTools_PutDTAInfo den Namen des Auftraggebers der Zahlung verändern und haben dazu 70 Zeichen zur Verfügung.

5) Beträge werden immer als Zeichenkette in Cent dargestellt. Es gibt kein Dezimalkomma oder Tausender-Separatoren. Beispiel: 1.342,76 € wird als 134276 dargestellt.

6) Für Lastschriften ist hier zwingend die Gläubigeridentifikation (CI) des Zahlungsempfängers zu übergeben. Diese muss bei der Deutschen Bundesbank beantragt werden (www.bundesbank.de). Zum Testen kann die CI DE98ZZZ09999999999 verwendet werden.

7) Innerhalb der Datensatzbeschreibung für DTA-Dateien stehen grundsätzlich keine Felder für Mandatsinformationen zur Verfügung. Dies kann aber mit einer Hilfslösung umgangen werden.

   Mandatsinformationen werden erkannt, wenn Sie folgendermaßen deklariert werden:

   **//MID** Mandatsreferenz

   Über das Kürzel //MID (Groß- oder Kleinschreibung ist egal) wird erkannt, dass der folgende Text eine Mandatsreferenz darstellt. Der auf das Kürzel folgende Text wird daher als Mandat übernommen.

   **//MDAT** 17.04.2011

   Über das Kürzel //MDAT (Groß- oder Kleinschreibung ist egal) wird erkannt, dass sich dahinter das Datum verbirgt, ab dem das Mandat gültig ist. Das Datum (für das Beispiel 17.04.2011) kann in folgenden Formaten eingegeben werden.

   | | |
   |---|---|
   | TT.MM.JJJJ | 17.04.2011 |
   | TT.MM.JJ | 17.04.11 |
   | TTMMJJJJ | 17042011 |
   | TTMMJJ | 170411 |

   **//SEQ** FRST

Über das Kürzel //SEQ können Sie die Sequenz, den Rhythmus von Lastschriften angeben. Gültige Werte sind dabei:

| | |
|---|---|
| FRST | für eine erstmalige Einreichung |
| RCUR | für eine wiederkehrende Einreichung (wohl der Normalfall) |
| OOFF | für eine einmalige Einreichung |
| FNAL | für eine letztmalige Einreichung |

**//LS** Sequenz&MDat&MId

Über dieses Kürzel ist es möglich, die für Lastschriften relevanten, zusätzlichen Informationen in einer Zeile zu übergeben. Die drei Werte sind durch das Zeichen **&** (kaufmännisches Und) getrennt. Dabei sind die Werte folgendermaßen zu übergeben:

**Sequenz**

| | |
|---|---|
| F | Buchstabe F für First (FRST) |
| R | Buchstabe R für Wiederkehrend (RCUR) |
| O | Buchstabe O für Einmalig (OOFF) |
| L | Buchstabe L für Letztmalig (FNAL) |

**Mandatsdatum**

Die Übergabe des Datums des Mandats (in diesem Beispiel der 17. April 2011) kann in folgenden Formaten übergeben werden.

| | |
|---|---|
| TT.MM.JJJJ | 17.04.2011 |
| TT.MM.JJ | 17.04.11 |
| TTMMJJJJ | 17042011 |
| TTMMJJ | 170411 |

**Mandats-Id**

Die Übergabe der Mandats-Id erfolgt im letzten Feld. Um für die Mandats-Id noch genügend Platz zu haben, empfiehlt es sich beim Mandatsdatum ein möglichst kurzes Format zu verwenden.

Werden einzelne Werte nicht übergeben, so ist trotzdem das kaufmännische Und (&) zu setzen. In diesem Fall werden dann die Vorgabewerte (Ersatzwerte) aus der INI-Datei verwendet.

Beispiele:

**//LS**F&170411&Mandat123
**//LS**&170411&Mandat123
**//LS**R&&Mandat123

Die Werte können in folgende Felder einer DTA-Datei eingeben werden:

Verwendungszweckzeilen 1 bis 14
2. Feld für den Empfänger
2. Feld für den Auftraggeber

Ein beispielhafter Eintrag innerhalb der Empfängerdaten könnte für ein Mandat dann folgendermaßen aussehen:

Beispiel für Sonderinformationen:

//MID Referenz
//MDAT 17032011
//SEQ FRST

Nachdem die Mandatsinformationen (Sonderinformationen) erkannt wurden, werden diese aus den entsprechenden Feldern der DTA-Datei entfernt.

Wenn aus der DTA-Datei keine Mandatsinformationen ausgelesen werden können, so wird das in der Struktur angegebene Mandatskürzel und das Mandatsdatum aus der Struktur verwendet. Um das Mandatskürzel eindeutig festzulegen, beachten Sie bitte den Erläuterungspunkt 8).

8) Mit dem Wert „AutoMandat“ legen Sie fest, ob und in welcher Weise eine Nummerierung des Mandats (aufgebaut auf dem Mandatskürzel) erfolgt. Es stehen für den Wert „AutoMandat“ folgende Werte zur Verfügung:

   0 Es erfolgt keine Durchnummerierung der Mandatskürzel

   1 Es erfolgt eine Durchnummerierung der Mandatskürzel (z.B. TestMandat123). Das so künstlich erzeugte Mandat wird aber nur verwendet, wenn nicht bereits ein Mandat vorhanden ist.

   2 Es erfolgt eine Durchnummerierung der Mandatskürzel (z.B. TestMandat123). Das so künstlich erzeugte Mandat wird aber auch verwendet, wenn nicht bereits ein Mandat vorhanden ist. Das vorhandene Mandat wird damit überschrieben.

   Der Wert „MandatStartNr“ legt fest, ab welchem Wert die Durchnummerierung beginnt.

9) Für jeden Sammler innerhalb einer XML-Datei ist eine eindeutige Id zu verwenden. Im Rahmen der Erstellung der XML-Datei wird diese Id automatisch erstellt.

   Sie können jedoch hier ein bis zu 10stelliges Kürzel vergeben, das dann in die Bildung der Id mit einfließt. Wenn Sie kein Kürzel vergeben, so erfolgt die Bildung der Id mit voreingestellten Werten.

10) Für jeden Datensatz innerhalb einer XML-Datei ist eine eindeutige EndToEnd-Id zu verwenden. Im Rahmen der Erstellung der XML-Datei wird diese EndToEnd-Id automatisch erstellt.

    Sie können jedoch hier ein bis zu 10stelliges Kürzel vergeben, das dann in die Bildung der Id mit einfließt. Wenn Sie kein Kürzel vergeben, so erfolgt die Bildung der EndToEnd-Id mit voreingestellten Werten.

11) Das Ausführungsdatum muss für Lastschriften in der Zukunft (lt. SEPA Rulebook) liegen. Für Überweisungen darf das Datum nicht in der Vergangenheit liegen.

    Wenn das Ausführungsdatum ungültig ist, so wird durch die API ein gültiges Ausführungsdatum vergeben.

12) Mit der Belegung der Variablen „B2B“ wird die Art der Lastschrift gesteuert. folgende Werte sind möglich:

    0 Normale CORE-Lastschrift mit den Einreichungsfristen 3 oder 6 Tage.
    1 Firmenlastschrift B2B. Dies entspricht dem bisherigen Abbuchungsverfahren.
    2 Lastschrift (COR1) mit verkürzten Einreichungsfristen von 2 Tagen. Die verkürzte Einreichungsfrist gilt für erstmalige und wiederkehrende (usw.) Lastschriften gleichermaßen.

    Die Belegung mit dem Wert 2 darf nur erfolgen, wenn mit der Funktion SepaTools_SetVersionUndLand die Version mindestens auf 2.7 gesetzt wurde. Wurde diese Mindestversion nicht gesetzt, so wird im Falle der Anlieferung des Wertes 2 dieser Wert intern auf den Wert 0 gesetzt.

    Der Lastschrifttyp COR1 muss zwischen den Kreditinstituten bilateral vereinbart worden sein. In Deutschland und in Österreich ist dies der Fall.

## *28.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Es stehen weitere Daten zur Verfügung. Ein wiederholter Aufruf der Funktion (mit Parameter First=0) ist erforderlich. |
| -1 | Es stehen keine Informationen über logische Dateien einer DTA-Datei zur Verfügung. Wahrscheinlich wurde die Funktion SepaTools_ReadDTA noch nicht aufgerufen. |
| -2 | In der physikalischen DTA-Datei waren keine logischen DTA-Dateien enthalten. Bitte prüfen Sie die DTA-Datei. |
| -3 | Der übergebene Wert für LogDat liegt nicht in dem Bereich zu dem Daten zur Verfügung stehen. Dieser Rückgabecode kann eigentlich nur auftreten, wenn Sie den Wert für LogDat verändert haben. |
| -4 | Es ist ein logischer Speicherfehler aufgetreten (sollte nie vorkommen). Bitte informieren Sie den Hersteller. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 29 SepaTools_WriteDTAtoXML

## *29.1 Zweck der Funktion*

Mit dieser Funktion wird aus den bisher erzeugten temporären Dateien die XML-Datei erzeugt. Im Laufe der Erzeugung der XML-Datei werden die Datensätze sortiert und ggfls. gruppiert. Im Rahmen der Gruppierung werden auch die Werte für die PmInfoId (Sammlerkennung) erzeugt (siehe Kapitel 28.3 auf Seite 95.

## *29.2 Funktionsaufruf*

**int SepaTools_WriteDTAtoXML(CreateStruct *XMLCreateStruct)**

## *29.3 Parameter*

***CreateStruct*** Hier wird ein Pointer auf die Struktur CreateStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für SEPA-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

```
struct    XMLCreateStruct

{   char    ExportPfad[255+1]      Laufwerk und Pfad für die Ausgabe der XML-Datei.
    int     Lastschrift            Überweisung (=0) oder Lastschrift (=1).
    char    XMLName[35+1]          Der Name der zu erzeugenden XML-Datei 1).
    char    MsgId[35+1]            Message-Id (Kennung) für die XML-Datei 2).
    char    EinreicherName[70+1]   Name des Einreichers der Datei (mind. 3 Zeichen).
}
```

Zusätzliche Erklärungen:

1) Unabhängig von der übergebenen oder nicht übergebenen Dateinamenserweiterung wird die Dateinamenserweiterung immer auf .xml gesetzt.

   Wird hier ein Leerstring oder der Wert Dummy.xml übergeben, so verwaltet die API den Dateienamen intern. Dies ist erforderlich, wenn die XML-Datei als Byte-Array im Speicher ausgegeben werden soll. Siehe hierzu Kapitel 14.3 auf der der Seite 54.

2) Die Message-Id soll eine möglichst eindeutige Identifizierung der XML-Datei sein. Wenn Sie hier einen Leerstring übergeben, so wird intern automatisch eine Message-Id gebildet und in der Struktur zurückgegeben.

## *29.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich<br><br>**Hinweis:**<br><br>**Die Funktion liefert auch den Rückgabewert 0, wenn z. B. alle Datensätze fachlich fehlerhaft waren. Dies kann passieren, wenn als Auftraggeber-Bankleitzahl eine BLZ eingetragen ist, deren BIC über SEPA nicht erreichbar ist. Damit sind alle Datensätze fachlich fehlerhaft, obwohl der technische Ablauf in Ordnung war.**<br><br>**Über den mehrfachen Aufruf der Funktion GetDTAProtokoll kann das nachgewiesen werden.** |
| -1 | Es wurde ein formal unrichtiger Pfad (Länge < 2 Zeichen) für die Exportdatei übergeben. |
| -2 | Das angegebene Verzeichnis für die Exportdatei existiert nicht. |
| -3 | Der Datenträger, der im Pfadnamen der Exportdatei angegeben wurde ist schreibgeschützt. Die Datei kann nicht geschrieben werden. |
| -4 | Der Name des Einreichers der Daten wurde zu kurz (< 3 Zeichen) angegeben. |
| -6 | Der Pfadname für die Exportdatei ist zu lang (>200 Zeichen). |
| -7 | Das angegebene Laufwerk im Pfad für die Exportdatei ist nicht bereit (möglicherweise kein Datenträger eingelegt). |
| -8 | Der Name für die XML-Datei wurde zu kurz (< 3 Zeichen) angegeben. |
| -9 | Es stehen keine Informationen über logische Dateien einer DTA-Datei zur Verfügung. Wahrscheinlich wurde die Funktion SepaTools_ReadDTA noch nicht aufgerufen. |
| -10 | In der physikalischen DTA-Datei waren keine logischen DTA-Dateien enthalten. Bitte prüfen Sie die DTA-Datei. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 30 SepaTools_GetDTAProtokoll

## *30.1 Zweck der Funktion*

Mit dieser Funktion können Sie zum einen die fehlerfrei umgesetzten Datensätze und zum anderen die fehlerhaften Datensätze auslesen und ggfls. protokollieren.

Die Funktion muss dazu mehrfach aufgerufen werden, bis der Rückgabewert einen Wert<>0 enthält. Über den Parameter Fehler steuern Sie, ob die gültigen Datensätze oder die fehlerbehafteten Datensätze ausgegeben werden.

## *30.2 Funktionsaufruf*

**int SepaTools_GetDTAProtokoll(ReadStruct *XMLReadStruct, int Fehler, int *First)**

## *30.3 Parameter*

***RadStruct*** Diese Struktur wird bei jedem Aufruf dieser Funktion mit den Daten des Zahlungsverkehrsauftrages gefüllt.

Die Struktur ist nachfolgend dargestellt.

***Fehler*** Setzen Sie den Parameter „Fehler“ auf 0, wenn Sie die fehlerfreien Datensätze auslesen wollen. Wird der Parameter „Fehler“ auf 1 gesetzt, so werden die fehlerhaften, nicht umgesetzten Datensätze ausgegeben.

***First*** Setzen Sie bitte diesen Wert beim ersten Aufruf dieser Funktion auf den Wert 1. Damit werden Initialisierungswerte gesetzt.

Der Wert wird von der API im Rahmen des ersten Funktionsaufrufes auf 0 gesetzt. Dieser Wert (0) ist dann für weitere Funktionsaufrufe zu übergeben (bis der Rückgabewert<>0 ist).

```
struct   XMLReadStruct

{   char    PmInfoId[35+1]          PaymentInfoId – Kennzeichnung des Auftrags
    char    AusfDatum[8+1]          Ausführungsdatum der Zahlungen im Format TTMMJJJJ.
    char    AuftragName[70+1]       Name des Auftraggebers.
    char    AuftragBIC[11+1]        BIC des Auftraggebers.
    char    AuftragIBAN[35+1]       IBAN des Auftraggebers.
    char    AuftragAbwName[70+1]    Optional einen abweichenden Namen des Auftraggebers.
    char    AuftragCI[35+1]         CI des Auftraggebers, nur bei Lastschriften.
    char    EmpfName[70+1]          Name des Empfängers der Zahlung.
    char    EmpfBIC[11+1]           BIC des Empfängers.
    char    EmpfIBAN[35+1]          IBAN des Empfängers.
    char    EmpfAbwName[70+1]       Optional ein abweichender Empfängername
    char    Purpose[4+1]            Optionale Angabe eines Zwecks der Zahlung.
    char    Betrag[12+1]            Betrag der Zahlung in Cent.
    char    EndToEndId[35+1]        Kennung der Einzelzahlung.
    char    MandatId[35+1]          Mandatskennung (nur) für Lastschriften.
    char    MandatDatum[8+1]        Datum des Mandats im Format TTMMJJJJ.
    char    SequenceType[4+1]       Sequence der Lastschrift.
    int     B2B                     B2B-Lastschrift (=1), norm. Lastschrift (=0), COR1 (=2).
```

| | | |
|---|---|---|
| int | SammlerAnzahl | Anzahl der Zahlungsaufträge in diesem Sammler. |
| char | SammlerSumme[12+1] | Summe der Beträge dieses Sammlers in Cent. |
| char | Zweck1[70+1] | Verwendungszweck Zeile 1 |
| char | Zweck2[70+1] | Verwendungszweck Zeile 2 |
| int | Hinweis[30] | Ein Feld mit bis zu 30 Hinweiskennziffern.[1)] |

}

Zusätzliche Erklärungen:

1) Bei dieser Verwendung der Struktur XMLReadStruct wird nur der erste Wert innerhalb des 30stelligen Hinweis-Feldes entweder mit >=0 (fehlerfrei) oder einem Fehlerwert aus folgender Liste (Detail-Fehlercodes) gefüllt.

## *30.4 Detail-Fehlercodes*

**Hinweis:**

Bei Returncodes >=0 (keine Fehler), werden im Hinweisfeld auch zusätzliche Informationen gespeichert. **Diese Informationen sind bitweise abgelegt**.

**Die Bitbelegungen für den Wert >= 0 im Hinweisfeld haben folgende Bedeutung:**

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| 1 | Die errechnete IBAN für den Auftraggeber kann verwendet werden. Sie ist aber nicht eindeutig. Eine Rückfrage beim Kunden wird empfohlen. |
| 2 | Die errechnete IBAN für den Empfänger der Zahlung kann verwendet werden. Sie ist aber nicht eindeutig. Eine Rückfrage beim Kunden wird empfohlen. |

**Ab hier sind die Fehlercodes wieder als negative Integer-Werte auszuwerten.**

| | |
|---|---|
| -104 | Der Betrag der Zahlung ist Null. |
| -105 | Die Bank des Zahlungsempfängers kann über SEPA nicht erreicht werden. |
| -110 | Für die Kombination aus Bankleitzahl und Kontonummer des Zahlungsempfängers kann keine IBAN berechnet werden. |
| -111 | Aus der Kombination von Bankleitzahl und Kontonummer des Zahlungsempfängers kann keine eindeutige IBAN berechnet werden. |
| -112 | Die Kontonummer des Zahlungsempfängers ist ungültig (Prüfziffer falsch). |
| -113 | Die Bankleitzahl des Empfängers ist zur Löschung vorgemerkt. |
| -116 | Die zur Mandatsänderung übergebene ursprüngliche IBAN ist ungültig. |
| -117 | Die Kennung SMNDA darf nur in Verbindung mit der Sequenz FRST verwendet werden. |
| -118 | Der ursprüngliche Name des Kreditors darf nicht mit dem aktuellen Namen identisch sein. |
| -119 | Das ursprüngliche Mandat darf nicht mit dem aktuellen Mandat identisch sein. |

-155 Die Bank des Auftraggebers kann über SEPA nicht erreicht werden.
-159 Die übergebene CI (Gläubigeridentifikation) ist ungültig.
-165 Für die Kombination aus Bankleitzahl und Kontonummer des Auftraggebers der Zahlung kann keine IBAN berechnet werden.

-166 Aus der Kombination von Bankleitzahl und Kontonummer des Auftraggebers der Zahlung kann keine eindeutige IBAN berechnet werden.

-167 Die Kontonummer des Auftraggebers der Zahlung ist ungültig (Prüfziffer falsch).
-168 Die Bankleitzahl des Auftraggebers ist zur Löschung vorgemerkt.
-169 Der Wert der Häufigkeit der Lastschrift ist ungültig. Erlaubt sind folgende Werte:

| | |
|---|---|
| FRST | für eine erstmalige Einreichung |
| RCUR | für eine wiederkehrende Einreichung (wohl der Normalfall) |
| OOFF | für eine einmalige Einreichung |
| FNAL | für eine letztmalige Einreichung |

-173 Die zur Mandatsänderung übergebene ursprüngliche CI ist ungültig.

-174 Die ursprüngliche CI darf nicht mit der aktuellen CI identisch sein.

### *30.5 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Es stehen weitere Daten zur Verfügung. Ein wiederholter Aufruf der Funktion (mit Parameter First=0) ist erforderlich. |
| -1 | Es stehen keine Informationen über konvertierte DTA-Dateien zur Verfügung. Wahrscheinlich wurde die Funktion SepaTools_ReadDTA noch nicht aufgerufen. |
| -2 | In der physikalischen DTA-Datei waren keine logischen DTA-Dateien enthalten. Bitte prüfen Sie die DTA-Datei. |
| -3 | Die temporäre Datei konnte nicht zum Lesen geöffnet werden. |
| -4 | Sie sind am Ende der Daten angelangt. Es stehen keine weiteren Daten über mehr zur Verfügung. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 31 SepaTools_FreeDTAVars

## *31.1 Zweck der Funktion*

Die Funktion gibt die von der API für das Konvertieren der DTA-Dateien belegten Speicherbereiche wieder frei. Ebenso werden temporär erzeugte Dateien wieder gelöscht.

**int SepaTools_FreeDTAVars()**

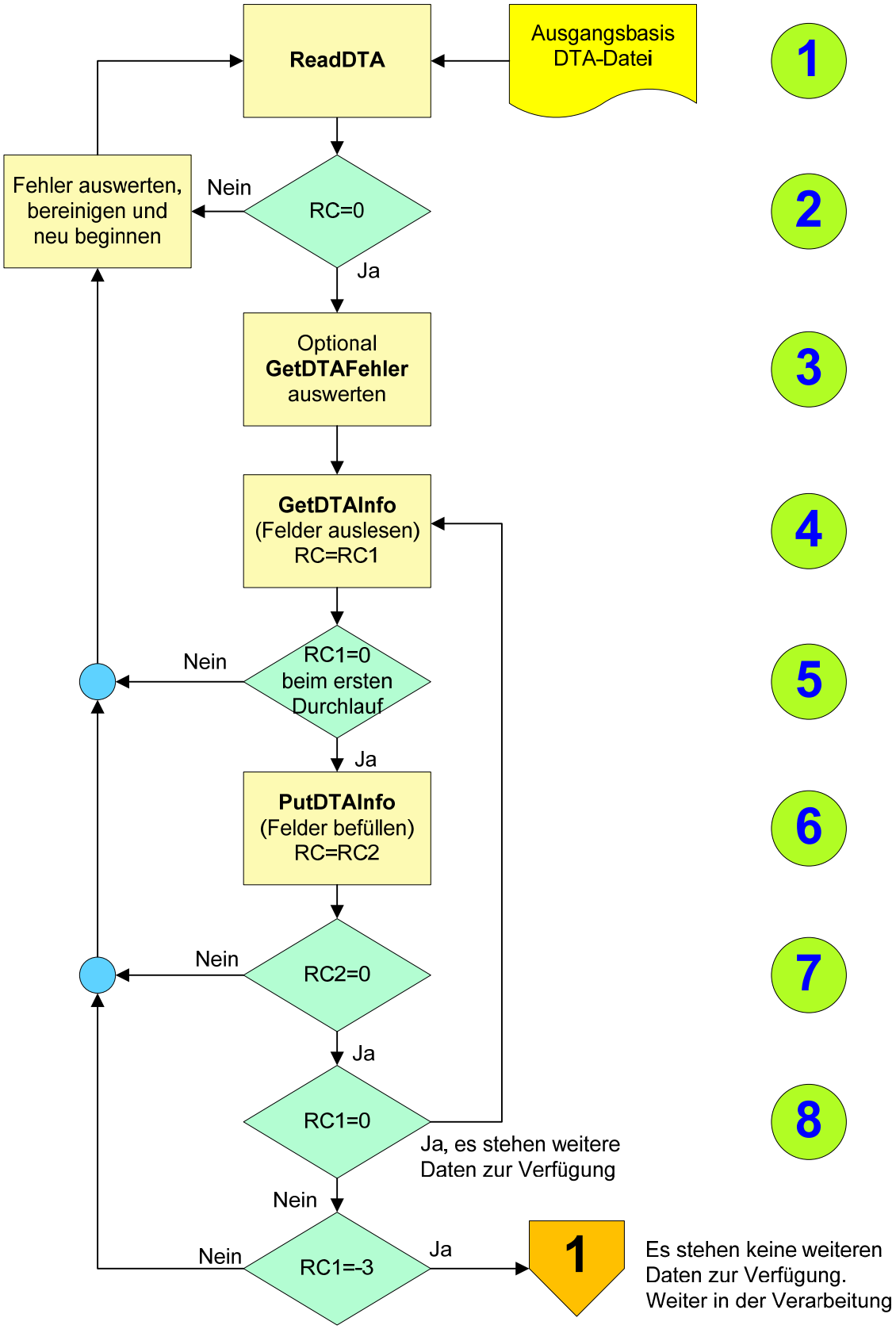
## *31.2 Parameter*

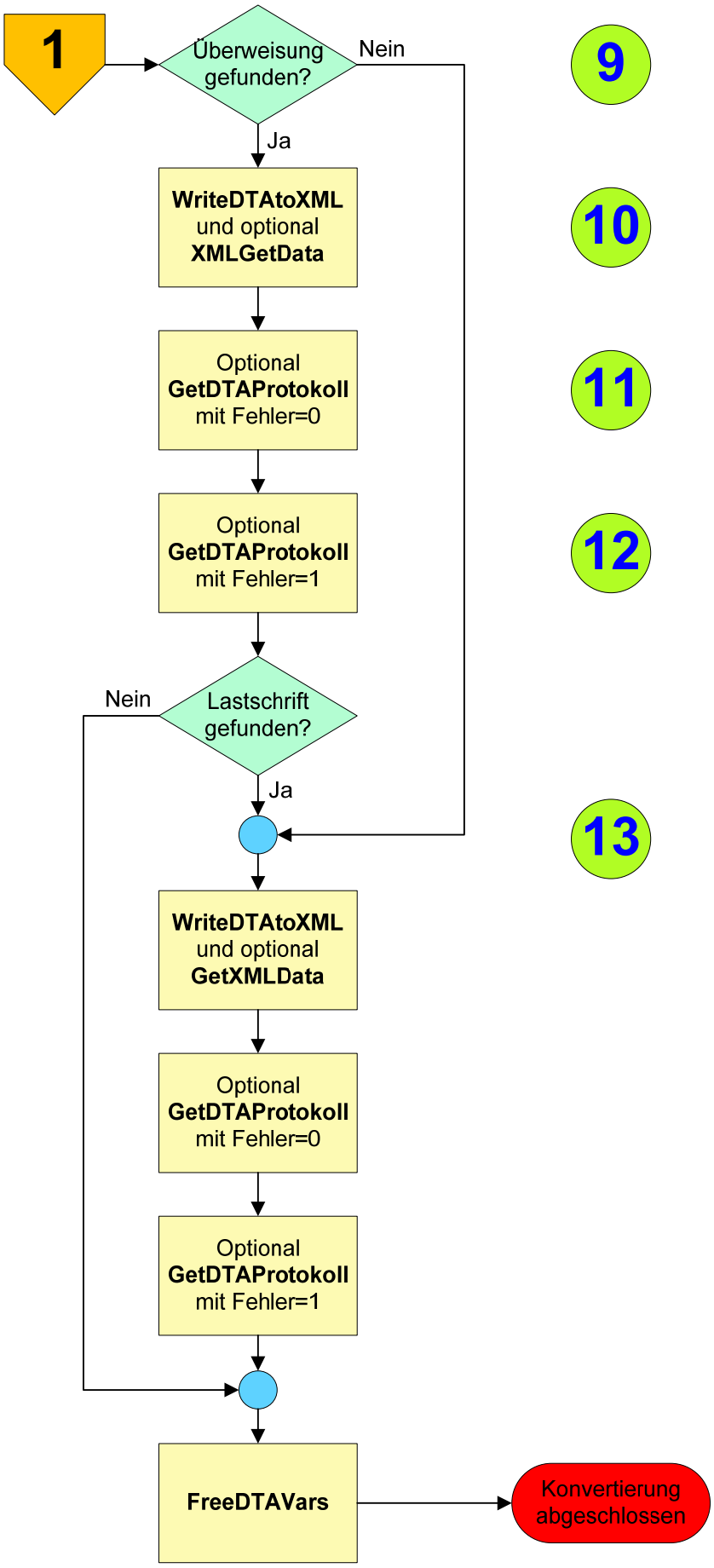
keine.

## *31.3 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

### *31.4 Ablaufdiagramm*




1
Überweisung gefunden?
Nein
Ja
WriteDTAtoXML und optional XMLGetData
Optional GetDTAProtokoll mit Fehler=0
Optional GetDTAProtokoll mit Fehler=1
Nein
Lastschrift gefunden?
Ja
WriteDTAtoXML und optional GetXMLData
Optional GetDTAProtokoll mit Fehler=0
Optional GetDTAProtokoll mit Fehler=1
FreeDTAVars
Konvertierung abgeschlossen
9
10
11
12
13

## *31.5 Beschreibung des Ablaufdiagramms*

Im Folgenden sind die Textziffern (grüner Kreis mit blauer Zahl) zum Ablaufdiagramm für die Konvertierung von DTA-Dateien in SEPA XML-Dateien beschrieben.

**1** Ausgangsbasis ist eine DTA-Datei. Es kann sich hierbei auch um eine sogenannte Multi-DTA Datei handeln, bei der sich in einer physikalischen Datei mehrere logische Dateien befinden.

Auch wenn es sich in der Regel nur um eine logische Datei in einer physikalischen Datei handelt, wird der Vollständigkeit halber immer von einer Multi-DTA Datei ausgegangen.

Werden beim Einlesen der DTA-Datei Fehler gefunden, die eine weitere Verarbeitung verhindern, so bricht die Funktion mit einem negativen Rückgabecode ab.

Werden fehlerhafte Datensätze gefunden, die eine weitere Verarbeitung der DTA-Datei nicht ausschließen (z.B. Betrag=0 oder Bankleitzahl nicht 8stellig), so werden nur diese fehlerhaften Datensätze von der weiteren Verarbeitung ausgeschlossen.

**2** Ein negativer Rückgabecode verhindert die weitere Verarbeitung.

**3** Wurden fehlerhafte Datensätze gefunden, die eine weitere Verarbeitung nicht ausschließen, so können diese betroffenen Datensätze optional durch einen wiederholten Aufruf der Funktion „GetDTAFehler“ abgerufen werden.

Die Fehler selbst können nicht geheilt werden. Die über die Funktion „GetDTAFehler“ erhaltenen Informationen dienen daher lediglich der Korrektur künftiger DTA-Dateien.

**4** Über die Funktion „GetDTAInfo“ werden Informationen der logischen Datei innerhalb der physikalischen DTA ausgelesen. Darin sind auch die Summen aus dem E-Satz (Anzahl Datensätze, Summe Beträge, Bankleitzahlen und Kontonummern) der DTA-Datei enthalten.

Bitte beachten Sie, dass es sich hier um die Beträge aus dem E-Satz handelt. Differenzen aufgrund von fehlerhaften Datensätzen sind hier nicht enthalten.

Diese Funktion sollte mehrfach aufgerufen werden, wenn die Möglichkeit besteht, dass sich in der physikalischen DTA-Datei mehr als eine logische DTA-Datei befindet.

**5** Da zumindest eine logische Datei vorhanden sein muss, wird bei einem erfolgreichen Aufruf dieser Funktion auch der Rückgabecode 0 zurückgegeben. Ist dies beim ersten Aufruf nicht der Fall, so ist ein Fehler aufgetreten, der die weitere Verarbeitung verhindert.

**6** Zumindest bei Lastschriften sollte die an die Funktion „PutDTAInfo“ zu übergebende Struktur mit weiteren Informationen angereichert werden. Wesentlich ist hierbei das Feld „Ausführungsdatum“, bei dem die Ausführungsfristen für Lastschriften zu beachten sind.

Bei Lastschriften wird geprüft, ob das Ausführungsdatum zumindest der Ausführungsfrist für wiederkehrende Lastschriften (aktuelles Tagesdatum+3 Tage unter Berücksichtigung der Target-Tage) entspricht. Ist das übergebene Ausführungsdatum kleiner als dieses Datum, so wird da so berechnete Datum verwendet.

**7** Bezüglich der Rückgabecodes beachten Sie bitte die Dokumentation.

**8** Wenn sich in der physikalischen DTA-Datei mehrere logische DTA-Dateien befinden, so ist der Aufruf der Sequenz „GetDTAInfo“ und „PutDTAInfo“ so lange zu wiederholen, bis der Rückgabewert der Funktion „GetDTAInfo“ einen Wert <>0 (-3 bedeutet keine weiteren Daten mehr vorhanden) aufweist.

**9** In der DTA-Datei können durchaus Überweisungen und Lastschriften gemeinsam enthalten sein. Da es sich in diesem Fall um unterschiedliche XML-Dateien handelt, muss die Ausgabe der XML-Datei bei dieser Annahme doppelt erfolgen.

**10** Über die Funktion „WriteDTAtoXML“ wird die eigentliche XML-Datei aus den zuvor gewonnenen Daten erzeugt.

In der zu übergebenden Struktur werden auch der Pfad und der Dateiname für die XML-Datei übergeben. Wird als Pfad und Dateiname ein Leerstring übergeben, so erfolgt die Ausgabe der XML-Datei nicht in eine Datei, sondern Byte-Array im Speicher.

Die Daten können dann optional über die Funktion „XMLGetData“ ausgelesen werden. Bitte beachten Sie die Dokumentation im Kapitel 0 ab Seite 84.

**11** Optional kann die Funktion „GetDTAProtokoll“ aufgerufen werden. Hier werden in der dokumentierten Struktur alle erfolgreich in die XML-Datei geschriebenen Datensätze zurückgeliefert. Übergeben Sie hierzu über den Parameter „Fehler“ den Wert 0.

**12** Um die fehlerhaften, nicht in die XML-Datei geschriebenen Datensätze zu protokollieren (z.B. Bank ist über SEPA nicht erreichbar), können Sie optional die Funktion „GetDTAProtokoll“ erneut aufrufen.

Bitte beachten Sie, dass dies nicht die gleichen Fehler sind, wie sie über die Funktion „GetDTAFehler“ ermittelt werden. Dort wurden technisch falsche Datensätze bereits aussortiert.

Über die Funktion „GetDTAFehler“ werden fachlich falsche Datensätze aussortiert, wie z.B. Bankleitzahl gelöscht, Bank über SEPA nicht erreichbar, Prüfziffer der Kontonummer falsch usw.

**13** Die in den Textziffern 10 bis 12 dargestellte Sequenz muss ggfls. ein zweites Mal aufgerufen werden.

# 32 DTAZV-Dateien in XML-Dateien umsetzen

Mit der API können DTAZV-Dateien (Deutscher Auslandszahlungsverkehr) in XML-Dateien umgesetzt werden. Damit können zumindest die Datensätze für den SEPA-Raum kostengünstiger weitergeleitet werden.

Die Vorgehensweise ist einfacher als bei DTA-Dateien, da es hier keine Lastschriften und keine Multi-DTA-Dateien gibt.

Da in einer DTAZV-Datei auch Datensätze enthalten sein können, die nicht in eine SEPA-Zahlung umgesetzt werden können, werden diese von der Umsetzung ausgenommen und wieder in einer (wenn auch reduzierten) DTAZV-Datei zur Verfügung gestellt.

Der grundsätzliche Ablauf stellt sich dabei folgenermaßen dar:

- Aufruf SepaTools_ConvertDTAZVtoXML — Die physikalische DTAZV-Datei wird eingelesen. Dabei werden SEPA-fähige Zahlungen in eine XML-Datei geschrieben.

  Zahlungen, die nicht SEPA-fähig sind (z. B. Währung nicht EUR, oder kein SEPA-Land) werden in eine zweite DTAZV-Datei geschrieben. Diee können dann wie bisher verarbeitet werden.

- Aufruf von SepaTools_GetDTAProtokoll — Über den wiederholten Aufruf dieser Funktion können Protokolldaten der erfolgreich und nicht erfolgreich geschriebenen Datensätze abgeholt werden.

  Es handelt sich hier um die gleiche Funktion, wie sie auch bei der Konvertierung von DTA-Dateien verwendet wird.

  Als fehlerfrei werden nur die Datensätze ausgegeben, die auch in die XML-Datei geschrieben wurden. Alle anderen Datensätze, die wieder in die DTAZV-Datei geschrieben wurden, werden als fehlerhafte Datensätze protokolliert.

# 33 SepaTools_ConvertDTAZVtoXML

## *33.1 Zweck der Funktion*

Die Funktion liest die DTAZV-Datei und konvertiert SEPA-fähige Datensätze zu SEPA-XML-Daten. Nicht SEPA-fähige Datensätze werden in eine zusätzliche DTAZV-Datei geschrieben.

## *33.2 Funktionsaufruf*

**int SepaTools_ConvertDTAZVtoXML(const AZVConvertStruct *AZVConvert)**

## *33.3 Parameter*

***AZVConvert*** Über diesen Parameter werden über einen Pointer auf einen Speicherbereich alle erforderlichen Variablen an die Funktion übergeben.

Die Struktur ist im Folgenden dargestellt.

struct AZVConvertStruct

| | | | |
|---|---|---|---|
| { | char | InputFile[255+1] | Der Pfad und der Dateiname für die DTAZV-Datei |
| | char | OutputFile[255+1] | Der Pfad und der Dateiname für die zu erzeug. XML-Datei |
| | char | AZVFile[255+1] | Der Pfad und der Dateiname für die zusätzl. DTAZV-Datei [1)] |
| | char | AusfDatum[8+1] | Das Ausführungsdatum für die SEPA-Zahlungen [2)] |
| | char | Kennung[10+1] | Ein Kürzel für die Kennungen in der XML-Datei [3)] |
| | char | Reserve[200] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Eine zusätzliche DTAZV-Datei wird nur erzeugt, wenn Datensätze gefunden werden, die nicht in SEPA-Datensätze umgesetzt werden können.

2) In der DTAZV-Datei ist ebenfalls bereits ein Ausführungsdatum enthalten. Wenn Sie hier ein Ausführungsdatum angeben, das größer als das Ausführungsdatum in der DTAZV-Datei ist, so wird dieses Datum verwendet.

   Geben Sie hier kein Datum, oder ein kleineres Datum an, so wird das Ausführungsdatum aus der DTAZV-Datei (InputFile) verwendet.

   Ist das Ausführungsdatum in der zu konvertierenden DTAZV-Datei ungültig und Sie haben hier kein Datum angegeben, so wird das Tagesdatum + 1 Tag verwendet.

3) Zur Kennzeichnung der einzelnen Datensätze in der XML-Datei werden Kennungen verwendet. Sie können hier ein bis zu 10stelliges Kürzel angeben, das zur Bildung dieser Kennungen verwendet wird. Die Kennung wird dann intern mit weiteren Informationen vervollständigt.

   Geben sie hier nichts an, so wird intern für das Kürzel der Wert „Kennung“ verwendet.

### *33.4 Returncodes*

0 Alles verlief erfolgreich

1 Alles verlief erfolgreich. Es konnten aber keine SEPA-fähigen Zahlungen gefunden werden. Die zusätzliche DTAZV-Datei wurde erstellt.

-1 Die angegebene DTAZV-Datei konnte nicht geöffnet werden, bzw. wurde nicht gefunden.

-2 Der Pfad für die Ausgabedatei (die zu erstellende XML-Datei) ist formal ungültig (kleiner als zwei Zeichen).

-3 Die Pfadangabe (mit Dateinamen) für die Ausgabedatei (XML-Datei) ist zu lang (länger als 200 Zeichen).

-4 Das Laufwerk im Pfad für die Ausgabedatei ist nicht bereit.

-5 Das Verzeichnis, das für die Ausgabedatei angegeben wurde existiert nicht.

-6 Der Datenträger, der für die Ausgabedatei angegeben wurde ist schreibgeschützt.

-7 Der Pfad für die AZV-Datei (die zusätzlich zu erstellende DTAZV-Datei) ist formal ungültig (kleiner als zwei Zeichen).

-8 Die Pfadangabe (mit Dateinamen) für die zusätzlich zu erstellende DTAZV-Datei ist zu lang (länger als 200 Zeichen).

-9 Das Laufwerk im Pfad für die zusätzlich zu erstellende DTAZV-Datei ist nicht bereit.

-10 Das Verzeichnis, das für die zusätzlich zu erstellende DTAZV-Datei angegeben wurde existiert nicht.

-11 Der Datenträger, der für die zusätzlich zu erstellende DTAZV-Datei angegeben wurde ist schreibgeschützt.

-21 Die temporäre Datei konnte (physikalisch) nicht erzeugt werden.

-22 Es handelt sich um keine gültige DTAZV-Datei (Eingangsdatei).

-23 In die temporäre Datei konnte nicht geschrieben werden (Schreibfehler).

-24 Die Bankleitzahl des Auftraggebers ist ungültig. Sie wurde im Datenbestand der Deutschen Bundesbank nicht gefunden.

-25 Die Kontonummer des Auftraggebers ist ungültig. Die Prüfziffernberechnung ist fehlgeschlagen.

-26 Die Bankverbindung des Auftraggebers kann nicht eindeutig in BIC und IBAN konvertiert werden. Bitte holen Sie nähere Informationen bei Ihrer Bank ein.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 34 SepaTools_GetDTAProtokoll

## *34.1 Zweck der Funktion*

Mit dieser Funktion können Sie zum einen die fehlerfrei umgesetzten Datensätze und zum anderen die fehlerhaften Datensätze auslesen und ggfls. protokollieren. Es handelt sich hierbei um die gleiche Funktion, die auch bei der Konvertierung von DTA-Dateien verwendet wird.

Die Funktion muss dazu mehrfach aufgerufen werden, bis der Rückgabewert einen Wert<>0 enthält. Über den Parameter Fehler steuern Sie, ob die gültigen Datensätze oder die fehlerbehafteten Datensätze ausgegeben werden.

## *34.2 Funktionsaufruf*

**int SepaTools_GetDTAProtokoll(ReadStruct *XMLReadStruct, int Fehler, int *First)**

## *34.3 Parameter*

Siehe bei der Originalbeschreibung dieser Funktion im Bereich der Umsetzung von DTA-Dateien.

## *34.4 Detail-Fehlercodes*

Siehe bei der Originalbeschreibung dieser Funktion im Bereich der Umsetzung von DTA-Dateien.

## *34.5 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Es stehen weitere Daten zur Verfügung. Ein wiederholter Aufruf der Funktion (mit Parameter First=0) ist erforderlich. |
| -3 | Die temporäre Datei konnte nicht zum Lesen geöffnet werden. |
| -4 | Sie sind am Ende der Datenbank angelangt. Es stehen keine weiteren Daten über mehr zur Verfügung. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 35 CSV-Dateien in XML-Dateien umsetzen

CSV-Dateien können mit der API in SEPA XML-Dateien umgesetzt werden. Mit dieser flexiblen Lösung, können aus Excel-Tabellen über die Ausgabe als CSV-Datei Zahlungsaufträge erzeugt werden.

Dabei lässt eine flexible Steuerung auch das Einsteuern von Informationen über die zu übergebende Struktur zu.

Der grundsätzliche Ablauf stellt sich dabei folgenermaßen dar:

- Aufruf SepaTools_ConvertCSVtoXML — Die CSV-Datei wird eingelesen. Dabei werden aus den einzelnen Feldern die Zahlungsaufträge gebildet und eine SEPA XML-Datei erzeugt.

- Aufruf von SepaTools_GetDTAProtokoll — Über den wiederholten Aufruf dieser Funktion können Protokolldaten der erfolgreich und nicht erfolgreich geschriebenen Datensätze abgeholt werden.

  Es handelt es sich auch hier wieder um die aus der DTA-Konvertierung bekannte Funktion.

# 36 SepaTools_ConvertCSVtoXML

## *36.1 Zweck der Funktion*

Die Funktion liest die CSV-Datei und konvertiert SEPA-fähige Datensätze zu SEPA-XML-Daten.

## *36.2 Funktionsaufruf*

**int SepaTools_ConvertCSVtoXML(const CSVConvertStruct *CSVConvert)**

## *36.3 Parameter*

***CSVConvert*** Über diesen Parameter werden über einen Pointer auf einen Speicherbereich alle erforderlichen Variablen an die Funktion übergeben.

Die Struktur ist im Folgenden dargestellt.

struct CSVConvertStruct

| | | | |
|---|---|---|---|
| { | char | InputFile[255+1] | Der Pfad und der Dateiname für die CSV-Datei. |
| | char | OutputFile[255+1] | Der Pfad und der Dateiname für die zu erzeug. XML-Datei |
| | char | AusfDatum[8+1] | Das Ausführungsdatum für die SEPA-Zahlungen [1)] |
| | char | Kennung[10+1] | Ein Kürzel für die Kennungen in der XML-Datei [2)] |
| | int | Lastschrift | Wert=0 für Überweisungen, Wert=1 für Lastschriften |
| | int | B2B | Wert für die Lastschriftart [3)] |
| | char | Einreicher[70+1] | Optionaler Name des Einreichers der XML-Datei [4)] |
| | char | AuftragName[70+1] | Optionaler Name des Auftraggebers der ZV-Aufträge [5)] |
| | char | AuftragBLZ[8+1] | Optionale Bankleitzahl des Auftraggebers [6)] |
| | char | AuftragKonto[11+1] | Optionale Kontonummer des Auftraggebers [7)] |
| | char | AuftragBIC[11+1] | Optionaler BIC des Auftraggebers [8)] |
| | char | AuftragIBAN[35+1] | Optionale IBAN des Auftraggebers [9)] |
| | char | CI[35+1] | Optionale CI des Auftraggebers [10)] |
| | char | Mandat[35+1] | Optional das Mandat, nur bei Lastschriften [11)] |
| | char | MandatDat[8+1] | Optional das Datum des Mandats, nur bei Lastschriften [12)] |
| | int | MandatStart | Opt. Startnummer für Durchnummerierung der Mandate [13)] |
| | int | CSVFeld[60] | Feld für die Positionen der Elemente in der CSV-Datei [14)] |
| | char | Trennzeichen[1+1] | Optional das Trennzeichen für die Elemente [15)] |
| | char | Grenze[1+1] | Optional das Zeichen für die Feldbegrenzung [16)] |
| | char | Tausend[1+1] | Optional das Zeichen für die Tausender-Separation [17)] |
| | char | Komma[1+1] | Opt. das Zeichen für die Kommadarstellung bei Beträgen [18)] |
| | int | HeaderZeilen | Opt. die Anzahl der Überschriftszeilen [19)] |
| | char | AusfZeit[8+1] | Optionale Ausführungszeit für Instant Zahlungen |
| | char | Reserve[187] | Reserve für spätere Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Geben Sie hier bitte das Ausführungsdatum für die Zahlungen im Format TTMMJJJJ an. Intern wird geprüft, ob das Ausführungsdatum für die gewählte Zahlung gültig ist. Dabei werden auch Wochenenden und Feiertage berücksichtigt.

Bei Lastschriften werden die Einreichungsfristen berücksichtigt. Liegt das Datum nicht weit genug in der Zukunft, so wird ein gültiges Datum berechnet.

2) Zur Kennzeichnung der einzelnen Datensätze in der XML-Datei werden Kennungen verwendet. Sie können hier ein bis zu 10stelliges Kürzel angeben, das zur Bildung dieser Kennungen verwendet wird. Die Kennung wird dann intern mit weiteren Informationen vervollständigt.

   Geben sie hier nichts an, so wird intern für das Kürzel der Wert „Kennung“ verwendet.

3) Geben Sie hier bitte an, welche Art von Lastschrift Sie erstellen wollen. Erlaubte Werte sind:

   0 Standard-Lastschrift mit den Einreichungsfristen von 3 Tagen bei wiederkehrenden Lastschriften und 6 Tagen bei erstmaligen Lastschriften.

   1 B2B - Firmenlastschrift mit einer Einreichungsfrist von 2 Tagen. Diese Lastschrift darf nur von Firmen (Nicht Verbrauchern) verwendet werden. Das Mandat muss bei der Bank des Zahlungspflichtigen hinterlegt sein.

   2 COR1 - Lastschrift mit verkürzter Einreichungsfrist von 2 Tagen. Diese Lastschriftart ist kein SEPA-Standard und muss zwischen den Kreditinstituten vereinbart sein. In Deutschland und in Österreich werden diese Vereinbarungen zum November 2013 geschlossen.

   Bitte fragen Sie bei Ihrer Bank nach, ob COR1 Lastschriften eingereicht werden können.

4) Geben Sie hier bitte optional den Namen des Einreichers der XML-Datei an. Grundsätzlich können in einer XML-Datei auch Zahlungsaufträge von unterschiedlichen Auftraggebern enthalten sein.

   Wenn Sie für den Einreicher der XML-Datei hier keinen Namen hinterlegen, so wird intern der erste gefundene Name eines Auftraggebers einer Zahlung auch für den Namen des Einreichers verwendet.

5) Grundsätzlich kann der Name des Auftraggebers der Zahlungen in einem Feld innerhalb der CSV-Datei mitgegeben werden. Ist der Name des Auftraggebers in der CSV-Datei nicht enthalten, so wird ersatzweise der angegebene Name verwendet.

6) Grundsätzlich kann die Funktion mit Bankleitzahlen und Kontonummern, sowie mit BIC und IBAN umgehen. Die Angabe dieser Werte in der CSV-Datei ist daher alternativ.

   Wird keine gültige Kombination aus BIC und IBAN gefunden (d.h. BIC **und** IBAN sind ungültig, beide Werte), so wird versucht aus der Bankleitzahl und der Kontonummer BIC und IBAN zu ermitteln. Schlägt auch dies fehl, so wird der Datensatz abgelehnt.

   Wenn in der CSV-Datei keine BLZ für den Auftraggeber mitgeliefert wird, so wird diese hier optional angegebene BLZ verwendet.

7) Für die Kontonummer des Auftraggebers gilt analog das für Punkt 6 Gesagte.

8) Für den BIC des Auftraggebers gilt analog das für Punkt 6 Gesagte.

9) Für die IBAN des Auftraggebers gilt analog das für Punkt 6 Gesagte.

10) Die CI (Gläubiger Identifikation) kann auch innerhalb der CSV-Datei mitgeliefert werden. Optional kann sie aber auch hier angegeben werden. Dann wird diese CI verwendet.

    Bitte achten Sie darauf, wenn Sie die CI in der CSV-Datei mitliefern, dass Sie dann auch den Namen des Auftraggebers ändern.

11) Das Mandat für eine Lastschrift sollte soweit möglich immer in der CSV-Datei mitgeliefert werden, da es sich um ein individuelles Mandat eines Zahlungspflichtigen handelt.

    Nur wenn dieses nicht beizubringen ist, wird hier über diesen Wert eine Mandats-Id erzeugt, die mit einem durchnummerierten Wert ergänzt wird.

12 Auch das Mandatsdatum sollte immer in der CSV-Datei angeliefert werden. Ist dies nicht möglich, so kann hier ein Datum im Format TTMMJJJJ übergeben werden. Dieses Datum wird dann für alle Mandate verwendet, bei denen kein gültiges Datum in der CSV-Datei mitgeliefert wurde.

13 Das ist die Startnummer, mit dem das wie unter Punkt 11 angegebene Mandat durchnummeriert ergänzt wird.

14 In diesem 30stelligen Feld legen Sie fest, an welcher Position der CSV-Datei die entsprechenden Feldinhalte zu finden sind. Dabei gelten Folgende Definitionen:

| | |
|---|---|
| CSVFeld[1] | Geben Sie hier bitte die Position des Namens des Auftraggebers in der CSV-Datei an (z.B. den Wert 3, wenn der Name des Auftraggebers der dritte Wert in der Zeile der CSV-Datei ist.<br><br>Wenn Sie den Namen des Auftraggebers nicht anliefern und aus der Struktur (siehe Punkt 5) entnehmen wollen, so belegen Sie bitte diesen Eintrag mit dem Wert 0. |
| CSVFeld[2] | Bankleitzahl des Auftraggebers. Ansonsten gilt das oben Gesagte. |
| CSVFeld[3] | Kontonummer des Auftraggebers. Ansonsten gilt das oben Gesagte. |
| CSVFeld[4] | BIC des Auftraggebers. Ansonsten gilt das oben Gesagte. |
| CSVFeld[5] | IBAN des Auftraggebers. Ansonsten gilt das oben Gesagte. |
| CSVFeld[6] | Name des Empfängers der Zahlung. Ansonsten gilt das oben Gesagte. |
| CSVFeld[7] | Bankleitzahl des Empfängers der Zahlung. Ansonsten gilt das oben Gesagte. |
| CSVFeld[8] | Kontonummer des Empfängers der Zahlung. Ansonsten gilt das oben Gesagte. |
| CSVFeld[9] | BIC des Empfängers der Zahlung. Ansonsten gilt das oben Gesagte. |
| CSVFeld[10] | IBAN des Empfängers der Zahlung. Ansonsten gilt das oben Gesagte. |
| CSVFeld[11] | Betrag der Zahlung. Ansonsten gilt das oben Gesagte. |
| CSVFeld[12] | Verwendungszweck der Zahlung. Ansonsten gilt das oben Gesagte. |

CSVFeld[13] CI (Gläubigeridentifikation) des Auftraggebers. Ansonsten gilt das oben Gesagte.

CSVFeld[14] Sequenz der Lastschrift. Wird hier kein Wert übergeben, so wird intern der Wert RCUR (wiederkehrende Lastschrift gesetzt).

CSVFeld[15] Mandat für die Lastschrift. Ansonsten gilt das oben Gesagte.

CSVFeld[16] Mandatsdatum. Ansonsten gilt das oben Gesagte. Das Datum kann im Format TTMMJJJJ oder TT.MM.JJJJ angeliefert werden.

CSVFeld[17] Nicht belegt.

CSVFeld[18] Geben sie hier optional die Position des Feldes für das ursprüngliche Mandat an.

CSVFeld[19] Geben sie hier optional die Position des Feldes für den ursprünglichen Namen des Kreditors an.

CSVFeld[20] Geben sie hier optional die Position des Feldes für die ursprüngliche CI des Kreditors an.

CSVFeld[21] Geben sie hier optional die Position des Feldes für die ursprüngliche IBAN des Debitors an.

CSVFeld[22] Geben sie hier optional die Position des Feldes für den Begriff SMNDA an.

CSVFeld[23] Geben Sie hier die optionale Position des Feldes für einen abweichenden Auftraggeber an.

CSVFeld[24] Geben Sie hier die optionale Position des Feldes für einen abweichenden Empfänger an.

CSVFeld[25] Geben Sie hier die optionale Position des Feldes für einen Purpose-Code an.

CSVFeld[26] Geben Sie hier die optionale Position des Feldes für das Ausführungsdatum an.

CSVFeld[27] Geben Sie hier die optionale Position des Feldes für die EndToEnd-ID an. Wenn Sie dieses Feld verwenden, so sind Sie selbst für die Eindeutigkeit der EndToEnd-ID pro Datensatz verantwortlich.

CSVFeld[28] bis CSVFeld[34] frei zur späteren Verwendung

CSVFeld[35] Geben Sie hier die optionale Position für die Länderkennung des Auftraggebers der Zahlung an.

CSVFeld[36] Geben Sie hier die optionale Position für die Adresszeile 1 des Auftraggebers der Zahlung an.

CSVFeld[37] Geben Sie hier die optionale Position für die Adresszeile 2 des Auftraggebers der Zahlung an.

CSVFeld[38] Geben Sie hier die optionale Position für die Länderkennung des Empfängers der Zahlung an.

CSVFeld[39] Geben Sie hier die optionale Position für die Adresszeile 1 des Empfängers der Zahlung an.

CSVFeld[40] Geben Sie hier die optionale Position für die Adresszeile 2 des Empfängers der Zahlung an.

**Hinweis:**

**Wenn Adressdaten geschrieben werden sollen, so ist das Länderkennzeichen ein Pflichtfeld. Ist dieses nicht vorhanden, so werden keine Adressdaten geschrieben.**

CSVFeld[41] Sie können hier eine optional die Position innerhalb der CSV-Datei für Uhrzeit die Uhrzeit der Ausführung der Instant-Überweisungen angeben.

Die Uhrzeit kann im Format HHMMSS oder mit Trennzeichen wie HH:MM.SS übergeben werden. Als Trennzeichen kann jedes nicht numerische Zeichen verwendet werden.

Wird keine Uhrzeit übergeben, so wird die optionale Uhrzeit in der Struktur ***CSVConvertStruct*** verwendet. Auch dort ist die Uhrzeit optional.

Die restlichen Einträge dienen der Reserve für künftige Erweiterungen und sollten mit 0 belegt sein.

15) Hier können Sie optional das Trennzeichen für die einzelnen Felder in der CSV-Datei angeben. Erlaubt sind folgende Zeichen:

**;** Der Strichpunkt
**,** Das Komma
**T** Der Wert T wird dann intern als Tabulatorzeichen gewertet.

Wird hier kein Wert übergeben, so wird intern der Strichpunkt (**;**) verwendet.

16) Hier können Sie optional angeben, ob und wenn ja welches Zeichen für die Begrenzung der Felder verwendet werden soll. Manchmal sind Felder in Anführungszeichen eingeschlossen.

Erlaubte Werte hierfür sind:

**„** Anführungszeichen
**'** Hochkomma
***/*** Schrägstrich

Auch wenn einzelne Felder nur in Begrenzungszeichen eingeschlossen sind, müssen Sie das Zeichen hier angeben, damit es dann bei der Verarbeitung ausgeschlossen wird.

Standardmäßig geht die Funktion davon aus, dass keine Feldbegrenzungen vorhanden sind.

17) Wenn Sie Beträge mit Tausenderseparatoren (z.B. 3.123.654,49) einlesen wollen, so können Sie hier optional das Zeichen für den Tausenderseparator angeben. Erfolgt hier keine Angabe, so wird intern der Punkt als Tausenderseparator angenommen.

18) Geben Sie hier optional das Zeichen für das Dezimalkomma an. Erlaubt sind das Komma und der Punkt (US-Darstellung). Wird hier nichts angegeben, so wird intern das Komma (deutsche Darstellung) verwendet.

19) Sie können optional angeben, ob die CSV-Datei Überschriftszeilen enthält, die überlesen werden sollen. Ein negativer Wert wird auch als 0 interpretiert. Ein Wert größer als 255 wird als 255 gewertet.

Bitte beachten:

In welcher Reihenfolge die einzelnen Felder in der CSV-Datei enthalten sind, ist nicht relevant. Ausgewertet werden maximal die ersten 50 Felder in der CSV-Datei. Hier müssen unabhängig von der Position die relevanten Daten untergebracht werden.

### *36.4 Returncodes*

0 Alles verlief erfolgreich

-1 Die Eingabedatei (XML-Datei) wurde nicht gefunden.

-2 Der Pfad für die Ausgabedatei (die zu erstellende XML-Datei) ist formal ungültig (kleiner als zwei Zeichen).

-3 Die Pfadangabe (mit Dateinamen) für die Ausgabedatei (XML-Datei) ist zu lang (länger als 200 Zeichen).

-4 Das Laufwerk im Pfad für die Ausgabedatei ist nicht bereit.

-5 Das Verzeichnis, das für die Ausgabedatei angegeben wurde existiert nicht.

-6 Der Datenträger, der für die Ausgabedatei angegeben wurde ist schreibgeschützt.

-7 Das Lesen der CSV-Datei war nicht erfolgreich.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 37 SepaTools_GetDTAProtokoll

## *37.1 Zweck der Funktion*

Mit dieser Funktion können Sie zum einen die fehlerfrei umgesetzten Datensätze und zum anderen die fehlerhaften Datensätze auslesen und ggfls. protokollieren. Es handelt sich hierbei um die gleiche Funktion, die auch bei der Konvertierung von DTA-Dateien verwendet wird.

Die Funktion muss dazu mehrfach aufgerufen werden, bis der Rückgabewert einen Wert<>0 enthält. Über den Parameter Fehler steuern Sie, ob die gültigen Datensätze oder die fehlerbehafteten Datensätze ausgegeben werden.

## *37.2 Funktionsaufruf*

**int SepaTools_GetDTAProtokoll(ReadStruct *XMLReadStruct, int Fehler, int *First)**

## *37.3 Parameter*

Siehe bei der Originalbeschreibung dieser Funktion im Bereich der Umsetzung von DTA-Dateien.

## *37.4 Detail-Fehlercodes*

Siehe bei der Originalbeschreibung dieser Funktion im Bereich der Umsetzung von DTA-Dateien.

## *37.5 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Es stehen weitere Daten zur Verfügung. Ein wiederholter Aufruf der Funktion (mit Parameter First=0) ist erforderlich. |
| -3 | Die temporäre Datei konnte nicht zum Lesen geöffnet werden. |
| -4 | Sie sind am Ende der Datenbank angelangt. Es stehen keine weiteren Daten über mehr zur Verfügung. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

## *37.6 Beispiel für die CSV-Konvertierung*

An folgender einfachen Beispieldatei (Überweisungen) soll die konkrete Umsetzung gezeigt werden.

```
Kundennummer1;600,25;Meierhofer Rita;Gehalt;Bemerkung1;26580070;732502200
Kundennummer2;1800,50;Altmannhofer Hans;Rechnung;Bemerkung2;60050101;2777939
```

Hinweis:

Die Felder Kundennummer und Bemerkung werden für die Konvertierung nicht benötigt, sind aber in dem Beispiel trotzdem vorhanden.

Ansonsten enthält die Datei den Betrag, den Empfängernamen, Verwendungszweck, Bankleitzahl und Kontonummer. Anstelle von BLZ und Kontonummer könnten auch BIC und IBAN vorhanden sein.

Die Auftraggeberdaten werden fest aus der Struktur übernommen.

### 37.6.1 Belegung der Struktur

struct CSVConvertStruct

| | **Feld/Variable** | | **Inhalt** |
|---|---|---|---|
| { | char | InputFile[255+1] | C:\Test\Input.csv |
| | char | OutputFile[255+1] | C:\Test\Überweisung.xml |
| | char | AusfDatum[8+1] | 25072013 |
| | char | Kennung[10+1] | Test |
| | int | Lastschrift | 0 |
| | int | B2B | 0 |
| | char | Einreicher[70+1] | |
| | char | AuftragName[70+1] | Berger Ulrich |
| | char | AuftragBLZ[8+1] | 74061813 |
| | char | AuftragKonto[11+1] | 70998 |
| | char | AuftragBIC[11+1] | |
| | char | AuftragIBAN[35+1] | |
| | char | CI[35+1] | |
| | char | Mandat[35+1] | |
| | char | MandatDat[8+1] | |
| | int | MandatStart | |
| | int | CSVFeld[30] | [0,0,0,0,0,3,6,7,0,0,2,4 . . . der Rest ist 0] |
| | char | Trennzeichen[1+1] | |
| | char | Grenze[1+1] | |
| | char | Tausend[1+1] | |
| | char | Komma[1+1] | |
| | char | Reserve[200] | |
| } | | | |

# 38 DTAZV-Dateien erzeugen

Mit der API können DTAZV-Dateien für den deutschen Auslandszahlungsverkehr erzeugt werden. Optional können aus AZV-Daten SEPA XML-Dateien erzeugt werden, wenn die Kriterien erfüllt sind.

Auf die Dokumentation der Deutschen Kreditwirtschaft (DK) wird ausdrücklich verwiesen. Diese wird auf der Webseite www.sepa-tools.de über den Menüpunkt „Sonstige Infos“ zur Verfügung gestellt.

Dazu sind drei Funktionen erforderlich, die nachfolgend dargestellt sind. Der logische Ablauf stellt sich folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateAZV | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_WriteAZV | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages übergeben. |
| … | |
| - Aufruf von SepaTools_CloseAZV | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |

Innerhalb dieser Funktionen finden umfangreiche Plausibilitätsprüfungen statt.

# 39 SepaTools_CreateAZV

## *39.1 Zweck der Funktion*

Mit dieser Funktion wird die Ausgabe von AZV-Dateien initialisiert. Des Weiteren sind hier die Daten des Einreichers der Zahlungen anzugeben. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

## *39.2 Funktionsaufruf*

**int SepaTools_CreateAZV( AZVCreateStruct *CreateStruct)**

## *39.3 Parameter*

***CreateStruct*** Hier wird ein Pointer auf die Struktur CreateStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für AZV-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Ebenfalls erfolgt eine Umsetzung in Großbuchstaben.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

struct AZVCreateStruct

| | | | |
|---|---|---|---|
| { | char | ExportPfad[255+1] | Laufwerk und Pfad für die Ausgabe der AZV-Datei. |
| | char | AZVName[35+1] | Der Name der zu erzeugenden AZV-Datei [1)]. |
| | char | EinreicherName1[35+1] | Name des Einreichers der Zahlungen (Q5) |
| | char | EinreicherName2[35+1] | Zus. Name des Einreichers der Zahlungen (optional) (Q5) |
| | char | EinreicherStrasse[35+1] | Anschrift des Einreichers der Zahlungen (optional) (Q5) |
| | char | EinreicherOrt[35+1] | Ort des Einreichers (optional) (Q5) |
| | char | EinreicherKonto[10+1] | Deutsche Kontonummer oder Kundennummer (Q4) |
| | char | EinreicherBLZ[8+1] | Bankleitzahl der dateiempfangenden Bank (Q3) |
| | char | AusfDatum[8+1] | Ausführungsdatum der Datei im Format TTMMJJJJ (Q8) |
| | int | TrySEPA | Optionale Prüfung auf SEPA-Fähigkeit [2)] |
| | char | MsgId[35+1] | Optionale Message-Id für eine optionale XML-Datei [3)] |
| | char | PmInfo[10+1] | Optionales Kürzel für die Bildung der Payment-Info-Id [4)] |
| | char | EndToEndId[10+1] | Optionales Kürzel für die Bildung der End-To-End-Id [5)] |
| | char | Reserve[500] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Wir hier ein Leerstring übergeben, so wird der Name für die AZV-Datei intern auf den Namen DTAZV (ohne Dateinamenerweiterung) gesetzt.

2) Wird hier der Wert 1 übergeben, so wird geprüft, ob die Zahlung auch als SEPA-Zahlung ausgeführt werden kann. Ist dies der Fall, so wird eine SEPA XML-Datei erzeugt und der entsprechende Datensatz in diese Datei geschrieben.

   Als Dateiname für die XML-Datei werden der gleiche Dateiname und der gleiche Exportpfad wie für die AZV-Datei verwendet. Es wird aber automatisch Die Dateinamenserweiterung .xml angefügt.

3) Die Message-Id soll eine möglichst eindeutige Identifizierung der XML-Datei sein. Diese wird nur benötigt, wenn die Variable TrySEPA auf den Wert 1 gesetzt wurde.

   Wenn Sie hier einen Leerstring übergeben, so wird intern automatisch eine Message-Id gebildet (als Kürzel wird intern der Wert AZVtoXML verwendet. Wird hier ein String mit bis zu 10 Zeichen übergeben, so werden diese 10 Zeichen als Kürzel für die Bildung der Message-Id verwendet.

4) Wenn die Variable TrySEPA auf den Wert 1 gesetzt ist, kann hier ein bis zu 10stelliges Kürzel für die Bildung der Payment-Info-Id übergeben werden. Wird hier ein Leerstring übergeben, so wird intern das Kürzel AZVtoXML verwendet.

5) Wenn die Variable TrySEPA auf den Wert 1 gesetzt ist, kann hier ein bis zu 10stelliges Kürzel für die Bildung der End-To-End-Id übergeben werden. Wird hier ein Leerstring übergeben, so wird intern das Kürzel AZVtoXML verwendet.

### *39.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Es wurde ein formal unrichtiger Pfad (Länge < 2 Zeichen) für die Exportdatei übergeben. |
| -2 | Das angegebene Verzeichnis für die Exportdatei existiert nicht. |
| -3 | Der Datenträger, der im Pfadnamen der Exportdatei angegeben wurde ist schreibgeschützt. Die Datei kann nicht geschrieben werden. |
| -6 | Der Pfadname für die Exportdatei ist zu lange (>200 Zeichen). |
| -7 | Das angegebene Laufwerk im Pfad für die Exportdatei ist nicht bereit (möglicherweise kein Datenträger eingelegt). |
| -8 | Der Name für die XML-Datei wurde zu kurz (< 3 Zeichen) angegeben. |
| -9 | Die Funktion SepaTools_CreateAZV wurde bereits aufgerufen. Vor einem weiteren Aufruf muss zuvor die Funktion SepaTools_CloseAZV aufgerufen werden.<br><br>Die Funktionalitäten zur Erzeugung einer DTAZV-Datei sind nicht „Multi-Tread“ fähig! |
| -10 | Die Datenbanken sind nicht aktuell. Sie benötigen die aktuellsten Datenbanken, die auch über die Länder- und Währungsinformationen verfügen. |

| | |
|---|---|
| -11 | Es wurde kein Auftraggeber-Name (Einreicher) übergeben. |
| -12 | Die Bankleitzahl der dateiempfangenden Bank ist ungültig. |
| -13 | Das übergebene Ausführungsdatum ist ungültig. Es muss sich im Bereich von „heute“ bis zu 15 Tagen später liegen. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 40 SepaTools_WriteAZV

## *40.1 Zweck der Funktion*

Über jeden Aufruf dieser Funktion wird genau ein Datensatz für einen Zahlungsauftrag übergeben. Es werden umfangreiche Plausibilitätsprüfungen durchgeführt. Werden bei der Plausibilitätsprüfung keine Fehler festgestellt, so wird der Datensatz in die DTAZV-Datei geschrieben.

Wurde in der CreateAZVStruct die Variable TrySEPA auf den Wert 1 gesetzt, so wird bei einem positiv geprüften Datensatz zusätzlich geprüft, ob dieser Datensatz auch als SEPA XML-Zahlung ausgeführt werden kann. Ist dies der Fall, so wird der Datensatz anstelle der DTAZV-Datei in die XML-Datei geschrieben.

## *40.2 Funktionsaufruf*

**int SepaTools_WriteAZV( AZVWriteStruct *WriteStruct)**

## *40.3 Parameter*

***WriteStruct*** Hier wird ein Pointer auf die Struktur WriteStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für AZV-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht.

struct AZVWriteStruct

| | | | |
|---|---|---|---|
| { | char | AuftragBLZ[8+1] | Bankleitzahl des Auftraggebers (T3) |
| | char | AuftragKonto[10+1] | Kontonummer des Auftraggebers (T4b) |
| | char | AuftragWhg[3+1] | ISO3-Code der Währung des Auftraggebers (T4a) |
| | char | AusfTermin[8+1] | Ausführungsdatum der Zahlung im Format TTMMJJJJ (T5) [1)] |
| | char | GebuehrBLZ[8+1] | Optionale BLZ für die Gebührenbelastung (T6) |
| | char | GebuehrKonto[10+1] | Optionale Kontonummer für die Gebührenbelastung (T7b) |
| | char | GebuehrWhg[3+1] | Optionale Währung für die Gebührenbelastung T17a) |
| | char | EmpfBIC[11+1] | BIC der Bank des Zahlungsempfängers (T8) [2] |
| | char | EmpfBankLand[2+1] | ISO2-Code des Landes der Bank des Zahl-.Empf. (T9a) [3)] |
| | char | EmpfBankName1[35+1] | Name der Bank des Zahlungsempfängers (T9b) [3)] |
| | char | EmpfBankName2[35+1] | Optional zusätz. Name der Bank des Zahlungsempf. (T9b) |
| | char | EmpfBankStrasse[35+1] | Optionale Adresse der Bank des Zahlungsempfängers (T9b) |
| | char | EmpfBankOrt[35+1] | Optionaler Ort der Bank des Zahlungsempfängers (T9b) |
| | char | EmpfLand[2+1] | ISO2-Code des Landes des Zahlungsempfängers (T10a) |
| | char | EmpfName1[35+1] | Name des Zahlungsempfängers (T10b) |
| | char | EmpfName2[35+1] | Optional zusätzlicher Name des Zahlungsempfängers (T10b) |
| | char | EmpfStrasse[35+1] | Optionale Adresse des Zahlungsempfängers (T10b) |
| | char | EmpfOrt[35+1] | Optionaler Ort des Zahlungsempfängers (T10b) |
| | char | Order1[35+1] | Ordervermerk 1 bei Scheckzahlungen (T11) |
| | char | Order2[35+1] | Ordervermerk 2 bei Scheckzahlungen (T11) |
| | char | IBAN[35+1] | IBAN oder Kontonummer des Zahlungsempfängers (T12) [4)] |
| | char | Whg[3+1] | Der ISO3-Code der Währung der Zahlung (T13) |
| | char | Betrag[15+1] | Der Betrag der Zahlung in “Cent” (T14) [5)] |

| | | |
|---|---|---|
| char | Zweck1[35+1] | Zeile 1 des Verwendungszwecks der Zahlung (T15) |
| char | Zweck2[35+1] | Zeile 2 des Verwendungszwecks der Zahlung (T15) |
| char | Zweck3[35+1] | Zeile 3 des Verwendungszwecks der Zahlung (T15) |
| char | Zweck4[35+1] | Zeile 4 des Verwendungszwecks der Zahlung (T15) |
| char | Weisung1[2+1] | Weisungsschlüssel 1 lt. DK-Dokumentation (T16) |
| char | Weisung2[2+1] | Weisungsschlüssel 2 lt. DK-Dokumentation (T17) |
| char | Weisung3[2+1] | Weisungsschlüssel 3 lt. DK-Dokumentation (T18) |
| char | Weisung4[2+1] | Weisungsschlüssel 4 lt. DK-Dokumentation (T19) |
| char | Zusatz[25+1] | Optionale Zusatzinformation (T20) |
| char | Entgelte[ 2+1] | Kennziffer für die Entgeltregelung (T21) 6) |
| char | ZVArt[2+1] | Kennzeichen der Zahlungsart lt. DK-Dokumentation (T22) |
| char | FreiText [27+1] | Optionaler variable Text für Auftraggeberabrechnung (T23) |
| char | Ansprechpartner[35+1] | Optionaler Ansprechpartner oder Telefonnummer (T24) |
| char | Reserve[500] | Reserve zur späteren Verwendung |

}

Zusätzliche Erklärungen:

1) Das Ausführungsdatum kann sich nur im Bereich von „heute“ bis zu den nächsten 15 Tagen bewegen. Wird kein Ausführungsdatum übergeben, so wird das Ausführungsdatum aus der Struktur AZVCreateStruct übernommen.

2) Handelt es sich bei der Bank des Zahlungsempfängers um eine deutsche Bank, so kann hier auch die Bankleitzahl übergeben werden. Die dann erforderlichen drei Schrägstriche (///) werden aber **intern, automatisch** eingefügt.

   Bei Scheckzahlungen ist dieses Feld nicht zu belegen. Wird es trotzdem belegt, so wird es intern wieder gelöscht.

3) Wurde im Feld „BIC“ kein Wert übergeben, so ist dieses Feld Pflicht. Bei Scheckzahlungen wird dieses Feld nicht belegt, bzw. intern wieder automatisch gelöscht.

4) Der vorangestellte Schrägstrich (lt. DK-Dokumentation) wird intern automatisch gesetzt. Bei Scheckzahlungen wird dieses Feld nicht belegt, bzw. intern wieder automatisch gelöscht.

5) Der Betrag der Zahlung wird immer in der kleinsten Stückelung angegeben.
   Für 2345,87 USD wird daher der Wert 234587 übergeben.

6) Angabe der Entgeltregelung lt. DK-Dokumentation.
   00 Entgelte zu Lasten Auftraggeber/fremde Entgelte zu Lasten Zahlungsempfänger
   01 Alle Entgelte und Auslagen zu Lasten des Auftraggebers.
   02 Alle Entgelte und Auslagen zu Lasten des Zahlungsempfängers.

## *40.4 Returncodes*

**Hinweis:**

Bei Returncodes >=0 (keine Fehler), werden im Rückgabewert auch zusätzliche Informationen gespeichert. **Diese Informationen sind bitweise abgelegt**.

**Die Bitbelegungen für den Wert >= 0 im Rückgabewert haben folgende Bedeutung:**

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |

| | |
|---|---|
| 1 | Die IBAN des Empfängers wurde geprüft. Diese ist möglicherweise falsch. Wenn es sich hier tatsächlich um eine IBAN handelt, sollte diese überprüft werden. |
| 2 | Die Länderkennung und die Währung des Zahlungsempfängers wurden gegenseitig geprüft. Die Währung passt nicht zu dieser Länderkennung. Bitte prüfen. |
| 4 | Es wurde kein Verwendungszweck übergeben. |
| 8 | Für diesen Datensatz wurde ein Eintrag in der optionalen SEPA XML-Datei erstellt. |

**Ab hier sind die Fehlercodes wieder als negative Integer-Werte auszuwerten.**

| | |
|---|---|
| -2 | Die Funktion SepaTools_CreateAZV wurde noch nicht aufgerufen. |
| -3 | Die DTAZV-Datei konnte nicht erzeugt werden. |
| -4 | Der Datensatz (T-Satz) konnte nicht in die DTAZV-Datei geschrieben werden. |
| -101 | Die Auftraggeber-BLZ ist ungültig. |
| -102 | Die Auftraggeber-Kontonummer ist ungültig (Prüfziffer). |
| -103 | Die Auftraggeber-Währung ist ungültig. |
| -104 | Das Ausführungsdatum ist ungültig. |
| -105 | Die BLZ für die Gebührenbelastung ist ungültig. |
| -106 | Die Kontonummer für die Gebührenbelastung ist ungültig. |
| -107 | Die Währung für die Gebührenbelastung ist ungültig. |
| -108 | IBAN oder Kontonummer des Zahlungsempfängers müssen angegeben werden. |
| -109 | Bei der gewählten Zahlungsart muss das Land angegeben werden (fehlender BIC). |
| -110 | Die Länderkennung für den Zahlungsempfänger ist ungültig. |
| -111 | Die Länderkennung für die Bank des Empfängers ist unbekannt. |
| -112 | Bei der gewählten Zahlungsart muss der Name der Empfängerbank angegeben werden (fehlender BIC). |
| -113 | Der Name des Zahlungsempfängers muss angegeben werden. |
| -114 | Die übergebene IBAN ist ungültig (bei Eilüberweisung). |

| | |
|---|---|
| -115 | Die Auftragswährung ist ungültig. |
| -116 | Der übergebene Betrag ist "0". |
| -117 | Es wurde keine Auftraggeber-Kontonummer übergeben. |
| -121 | Der Weisungsschlüssel 1 ist bei dieser Zahlungsart ungültig. |
| -122 | Der Weisungsschlüssel 2 ist bei dieser Zahlungsart ungültig. |
| -123 | Der Weisungsschlüssel 3 ist bei dieser Zahlungsart ungültig. |
| -124 | Der Weisungsschlüssel 4 ist bei dieser Zahlungsart ungültig. |
| -129 | Die Kombination der Weisungsschlüssel ist ungültig (siehe DK-Dokumentation). |
| -130 | Der Schlüssel für die Entgelte ist bei dieser Auftragsart ungültig. |
| -131 | Der Schlüssel für die Zahlungsart ist ungültig. |
| -151 | Bei der Zahlungsart "11" muss für die Empfängerbank ein BIC übergeben werden. |
| -161 | Bei der Zahlungsart „11“ muss der Weisungsschlüssel 1 (wenn vorhanden) 10, 11 oder 12 sein. |
| -162 | Bei der Zahlungsart „11“ muss der Weisungsschlüssel 2 (wenn vorhanden) 10, 11 oder 12 sein. |
| -163 | Bei der Zahlungsart „11“ muss der Weisungsschlüssel 3 (wenn vorhanden) 10, 11 oder 12 sein. |
| -164 | Bei der Zahlungsart „11“ muss der Weisungsschlüssel 4 (wenn vorhanden) 10, 11 oder 12 sein. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 41 SepaTools_CloseAZV

## *41.1 Zweck der Funktion*

Mit dieser Funktion wird die AZV-Datei geschlossen. Entsprechende Prüfsummen werden in der übergebenen Struktur übergeben.

## *41.2 Funktionsaufruf*

**int SepaTools_CloseAZV( AZVCloseStruct *CloseStruct)**

## *41.3 Parameter*

***CloseStruct*** Hier wird ein Pointer auf die Struktur CloseStruct übergeben. Die Struktur ist nachfolgend dargestellt. Die Struktur wird leer übergeben und von der API mit Informationen für die Applikation gefüllt.

struct AZVCloseStruct

| | | | |
|---|---|---|---|
| { | int | NumAZV | Anzahl fehlerfrei erzeugter DTAZV-Datensätze. |
| | int | NumSEPA | Anzahl der SEPA-Datensätze. |
| | int | NumInvalid | Anzahl ungültiger Datensätze. |
| | char | BetragAZV[12+1] | Summe der AZV-Beträge in “Cent”. |
| | char | BetragSEPA [12+1] | Summe der SEPA-Beträge in “Cent”. |
| | char | BetragInvalid[12+1] | Summe der Beträge der ungültigen Datensätze. |
| | char | Reserve[100] | Reserve zur späteren Verwendung. |
| } | | | |

## *41.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -2 | Die Funktion SepaTools_CreateAZV wurde noch nicht aufgerufen. |
| -4 | Der Datensatz (Z-Satz) konnte nicht in die DTAZV-Datei geschrieben werden. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 42 SepaTools_GetLandWhg

## *42.1 Zweck der Funktion*

Diese Funktion ermöglicht es, die Plausibilität von Länderkennungen und Währungscodes im Vorfeld zu überprüfen. Diese Funktion wird auch intern in der API zu Plausibilitätsprüfungen eingesetzt.

## *42.2 Funktionsaufruf*

**Int SepaTools_GetLandWhg(const char *Land,const char *Whg,**
**LandWhgStruct *LandWhg)**

## *42.3 Parameter*

***Land*** Bitte übergeben Sie hier den ISO2-Ländercode (z.B. DE, US, AT usw.), der überprüft werden soll. Wenn nur die Währung überprüft werden soll, so kann hier auch ein Leerstring übergeben werden.

***Whg*** Bitte übergeben Sie hier den ISO3-Währungscode (z.B. USD, EUR usw.), der überprüft werden soll. Wenn nur die Länderkennung überprüft werden soll, so kann hier auch ein Leerstring übergeben werden.

***LandWhg*** Dies ist ein Pointer auf die Struktur LandWhgStruct. In dieser Struktur werden die ermittelten Informationen wieder zurückgegeben.

Die Struktur ist nachfolgend dargestellt.

```
struct   LandWhgStruct

{   int      Info              Informationen über das Ergebnis des Funktionsaufrufs 1)
    char     LandISO2[2+1]     ISO2-Code des gefundenen Landes.
    char     LandISO3[3+1]     ISO3-Code des gefundenen Landes.
    int      LandNum           Numerischer Ländercode.
    char     LandName[40+1]    Name, Bezeichnung des Landes.
    char     WhgISO3[3+1]      ISO3-Code der gefundenen Währung.
    char     WhgName[40+1]     Name, Bezeichnung der Währung.
    char     Reserve[100]      Reserve zur späteren Verwendung
}
```

Zusätzliche Erklärungen:

1 Die Informationen werden im Feld „Info“ bitweise dargestellt. Die einzelnen Bit-Belegungen haben dabei folgende Bedeutung:

- 0 Keine besonderen Hinweise
- 1 Die übergebene Länderkennung ist ungültig oder wurde nicht gefunden.
- 2 Der übergebene Währungscode ist ungültig oder wurde nicht gefunden.
- 4 Währungscode und Länderkennung passen fachlich nicht zusammen.

## *42.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die Datenbanken konnten nicht geöffnet werden. |
| -10 | Die Datenbanken sind nicht aktuell. Sie benötigen die aktuellsten Datenbanken, die auch über die Länder- und Währungsinformationen verfügen. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 43 SepaTools_SetVersionUndLand

## *43.1 Zweck der Funktion*

Mit dieser Funktion können die XML-Version und das Einreicherland für die XML-Datei festgelegt werden. Als Einreicherland stehen Deutschland, Österreich und Schweiz zur Verfügung. Der Dateiaufbau unterscheidet sich geringfügig.

## *43.2 Funktionsaufruf*

**int SepaTools_SetVersionUndLand(int Version, intLand)**

## *43.3 Parameter*

***Version*** Siehe unten stehende Tabelle

**Legende für die Tabelle:**

| | |
|---|---|
| Gültig ab | Datum der Veröffentlichung |
| Version DK | Version Deutsche Kreditwirtschaft |
| Version RB | Rule Book Österreich |
| Version ST | Versionsnummer für SepaTools, die in die Funktion SetVersionUndLand eingesetzt werden muss. |
| Land | Land, für das die Version gilt |
| Formate | Formate für Überweisungen und Lastschriften |

| *Gültig ab* | *Version* | *Land* | *Formate* | *Bemerkung* |
|---|---|---|---|---|
| 22.11.2009 | DK 2.4<br>RB 3.2<br>Ver. ST=1 | DE<br>AT | pain.001.002.02<br>pain.008.002.01 | Keine Anmerkungen |
| 20.11.2010 | DK 2.5<br>RB 5<br>Ver. ST=2 | DE<br>AT | pain.001.003.03<br>pain.008.002.02 | Keine Anmerkungen |
| 17.11.2012 | DK 2.7<br>RB 6<br>Ver. ST=3 | DE<br>AT | pain.001.003.03<br>pain.008.002.02 | Es gibt keine Version 2.6, da diese technisch mit der Version 2.5 identisch war. |
| 17.11.2012 | RB 6<br>Ver ST=4 | AT | pain.001.002.03<br>pain.008.002.02 | Erzeugt eine XML-Datei nach Rulebook 6.0 für Österreich in einer besonderen Variante für die Deutsche Bank in Österreich. Dieser Wert kann nur gemeinsam mit dem Wert 1 für das Land (Österreich) verwendet werden.<br><br>Wird der Wert **4** gemeinsam mit dem Land Deutschland verwendet, so wird der Wert intern wieder auf den Wert **3** gesetzt. |
| 20.11.2016 | DK 3.0<br>RB 7 bis 9<br>Ver. ST=5 | DE<br>AT | pain.001.001.03<br>pain.008.001.02 | Diese muss von den Banken ab dem 20. November 2016 unterstützt werden. |
| 19.11.2017 | DK 3.1<br>RB 7 bis 9<br>Ver. ST=6 | DE<br>AT | pain.001.001.03<br>pain.008.001.02 | Nur marginale Änderungen, meist nur Klarstellungen in der Doku. |
| 17.11.2019 | DK 3.3 | DE | pain.001.001.03 | Die DK-Version 3.3. Diese muss von |

| Gültig ab | Version | Land | Formate | Bemerkung |
|---|---|---|---|---|
| | RB 7 bis 9<br>Ver. ST=7 | AT | pain.008.001.02 | den Banken ab dem 17. November 2019 unterstützt werden.<br><br>Hier wurden aber nur Präzisierungen in Bezug auf Instants-Zahlungen vorgenommen. Ansonsten ist die Version mit der Version 3.1 identisch. |
| 19.11.2023 | DK 3.7<br>RB offen<br>Ver. ST=8 | DE<br>AT | pain.001.001.03<br>pain.008.001.02<br>pain.001.001.09<br>pain.001.001.09.AXZ | Die Version pain.001.001.09 ist das Format für Überweisungen. Soweit die Banken das unterstützen, kann dieses Format schon heute verwendet werden.<br><br>Über dieses Format können auch Instant-Zahlungen mit der optionalen zusätzlichen Möglichkeit der Uhrzeit für den Ausführungszeitpunkt eingereicht werden.<br><br>Außerdem können über das Format pain.001.001.09 die künfigen Auslandsüberweisungen (verpflichtende ab November 2025) zur Ablösung des DTAZV-Formats eingereicht werden.<br><br>Um die für den AZV zusätzlich erforderlichen Felder prüfen zu können, steht die Validierungsdatei pain.001.001.09. AXZ zur Verfügung. |
| Nov. 2011 | ???<br>Ver. ST=50 | CH | pain.001.01.003<br>pain.008.001.02.ch.02<br>pain.008.001.02.chsdd.02 | Dieser Wert ist für den Einsatz der API in der Schweiz zu verwenden.<br><br>Obwohl es in der Schweiz derzeit noch keine unterschiedlichen Versionen gibt, ist die Angabe aus Kompatibilitätsgründen anzugeben. |
| Nov 2022 | ???<br>Ver. ST=51 | CH | pain.001.01.009<br>pain.008.001.02.ch.03<br>pain.008.001.02.chsdd.02 | Keine Anmerkungen. |
| Nov. 2025 | DK 3.9<br>Ver. ST=8 | DE<br>AT | pain.001.01.009<br>pain.008.01.008 | Aktuelle Versionen. |

***Land*** Um den unterschiedlichen Formatanforderungen für Deutschland Rechnung zu tragen, können Sie hier das Einreicherland definieren. Folgende Werte sind hier möglich:

Land_DE = 0 Deutschland

Land_AT = 1 Österreich

Land_CH = 2 Schweiz

Hinweis:

Wenn Sie diese Funktion nicht aufrufen, so ist standardmäßig ZKA-Version 3.3 für Deutschland eingestellt.

### *43.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Es wurde eine ungültige Version übergeben. |
| -2 | Es wurde eine ungültige Länderkennung übergeben. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 44 SepaTools_XMLLesenInit

## *44.1 Zweck der Funktion*

Mit dieser Funktion wird das Einlesen einer SEPA XML-Datei initialisiert. Es finden umfangreiche Plausibilitätsprüfungen statt. Anschließend wird die XML-Datei in eine temporäre Datei konvertiert, die dann über folgende Aufrufe von SepaTools_XMLLesen ausgelesen werden können.

## *44.2 Funktionsaufruf*

**int SepaTools_XMLLesenInit(const char *Pfad, const char *FN)**

## *44.3 Parameter*

| | |
|---|---|
| ***Pfad*** | Hier wird ein Pointer auf den Pfad übergeben, in dem sich die XML-Datei befindet. Der Pfad muss nullterminiert sein. |
| ***FN*** | Hier wird ein Pointer auf den Dateinamen der XML-Datei übergeben. Der Pfad muss nullterminiert sein. |

## *44.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Es wurde ein formal ungültiger Pfad (<2Zeichen) übergeben. |
| -2 | Das im Pfad angegebene Verzeichnis existiert nicht. |
| -3 | Die XML-Datei wurde nicht gefunden. |
| -5 | Die temporäre Datei konnte nicht erstellt werden. Bitte überprüfen Sie den über die Funktion SepaTools_Init übergebenen Pfad für die temporäre Datei (TempPath). |
| -6 | Der übergeben Pfad ist zu lang (>200 Zeichen). |
| -7 | Das in dem Pfad angegebene Laufwerk ist nicht bereit. |
| -8 | Der Name der XML-Datei ist zu kurz, <3 Zeichen. |
| -11 | Die XML-Datei konnte nicht gelesen werden. Möglicherweise ist die Datei für den verfügbaren Hauptspeicher zu groß. |
| -12 | Die XML-Datei ist keine gültige SEPA XML-Datei. Bitte überprüfen Sie die XML-Datei entsprechend dem SEPA Rulebook. |
| -13 | Die Group-Informationen konnten nicht in die temporäre Datei geschrieben werden. |
| -14 | In der XML-Datei wurden keine SEPA-Zahlungsverkehrsaufträge gefunden. |

| | |
|---|---|
| -15 | Die Zahlungsverkehrs-Informationen konnten nicht in die temporäre Datei geschrieben werden. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 45 SepaTools_XMLLesen

## *45.1 Zweck der Funktion*

Wenn vorher die Funktion SepaTools_XMLLesenInit aufgerufen wurde, so können mit dieser Funktion die einzelnen Datensätze aus der temporären Datei ausgelesen werden. Beim ersten Aufruf dieser Funktion muss der Wert *First* auf 1 gesetzt werden, ansonsten auf 0.

Wenn alle Datensätze ausgelesen wurden, wird die temporäre Datei automatisch geschlossen und gelöscht. Wenn Sie diese Funktion nicht bis zum kompletten Auslesen der temporären Datei aufrufen, müssen Sie zum Abschluss noch die Funktion SepaTools_XMLLesenClose aufrufen, damit die temporäre Datei wieder gelöscht wird.

## *45.2 Funktionsaufruf*

**int SepaTools_XMLLesen(const int *First, XMLGroupStruct *GroupStruct**
**XMLReadStruct *ReadStruct)**

## *45.3 Parameter*

***First*** Setzen Sie bitte diesen Wert beim ersten Aufruf dieser Funktion auf den Wert 1. Damit werden Initialisierungswerte gesetzt.

Bei weiteren nachfolgenden Aufrufen dieser Funktion ist dieser Wert dann auf 0 zu setzen.

***GroupStruct*** Diese Struktur wird nur bei dem ersten Aufruf dieser Funktion gefüllt. Der Inhalt bleibt erhalten, es sei denn, die Applikation löscht den Inhalt.

Die Struktur ist folgendermaßen aufgebaut:

```
struct    XMLGroupStruct

{   int      Lastschrift          Wert=1, wenn Lastschriften, ansonsten Wert=0.
    int      SummeAnzahl          Anzahl der Datensätze in der XML-Datei.
    char     SummeBetrag[12+1]    Gesamtsumme der Beträge in der XML-Datei in Cent.
    char     MsgId[35+1]          Message-Id der XML-Datei.
    char     ErstDatum[8+1]       Erstellungsdatum der XML-Datei im Format TTMMJJJJ.
    char     ErstZeit[8+1]        Erstellungszeit der XML-Datei im Format HH:MM.SS.
    char     EinreicherName[70+1] Name des Einreichers der XML-Datei
}
```

***RadStruct*** Diese Struktur wird bei jedem Aufruf dieser Funktion mit den Daten des Zahlungsverkehrsauftrages gefüllt.

Die Struktur ist folgendermaßen aufgebaut:

```
struct    XMLReadStruct

{   char     PmInfoId[35+1]       PaymentInfoId – Kennzeichnung des Auftrags
    char     AusfDatum[8+1]       Ausführungsdatum der Zahlungen im Format TTMMJJJJ.
    char     AuftragName[70+1]    Name des Auftraggebers.
```

| | | |
|---|---|---|
| char | AuftragBIC[11+1] | BIC des Auftraggebers. |
| char | AuftragIBAN[35+1] | IBAN des Auftraggebers. |
| char | AuftragAbwName[70+1] | Optional einen abweichenden Namen des Auftraggebers. |
| char | AuftragCI[35+1] | CI des Auftraggebers, nur bei Lastschriften. |
| char | EmpfName[70+1] | Name des Empfängers der Zahlung. |
| char | EmpfBIC[11+1] | BIC des Empfängers. |
| char | EmpfIBAN[35+1] | IBAN des Empfängers. |
| char | EmpfAbwName[70+1] | Optional ein abweichender Empfängername |
| char | Purpose[4+1] | Optionale Angabe eines Zwecks der Zahlung. |
| char | Betrag[12+1] | Betrag der Zahlung in Cent. |
| char | EndToEndId[35+1] | Kennung der Einzelzahlung. |
| char | MandatId[35+1] | Mandatskennung (nur) für Lastschriften. |
| char | MandatDatum[8+1] | Datum des Mandats im Format TTMMJJJJ. |
| char | SequenceType[4+1] | Sequence der Lastschrift. |
| int | B2B | B2B-Lastschrift (=1), normale Lastschrift (=0), COR1 (=2). |
| int | SammlerAnzahl | Anzahl der Zahlungsaufträge in diesem Sammler. |
| char | SammlerSumme[12+1] | Summe der Beträge dieses Sammlers in Cent. |
| char | Zweck1[70+1] | Verwendungszweck Zeile 1 |
| char | Zweck2[70+1] | Verwendungszweck Zeile 2 |
| int | Hinweis[30] | Ein Feld mit bis zu 30 Hinweiskennziffern (siehe später). |

}

### *45.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die temporäre Datei konnte nicht geöffnet werden. |
| -2 | Die Funktion SepaTools_XMLLesenInit wurde noch nicht aufgerufen. |
| -3 | Sie sind am Ende der temporären Datei angekommen. Es stehen keine weiteren Datensätze mehr zur Verfügung. |
| -4 | Das Auslesen des GroupHeaders war nicht erfolgreich. |
| -5 | Beim Lesen des Zahlungsauftrages ist ein Fehler aufgetreten. |

### *45.5 Hinweise*

Im Rahmen der Umsetzung der XML-Datei werden u.U. Hinweis-Kennziffern erzeugt. Diese Hinweise bedeuten nicht, dass der Vorgang abgebrochen werden muss. Sie zeigen lediglich Unstimmigkeiten innerhalb der XML-Datei auf.

In der Struktur XMLReadStruct befindet sich für jeden Datensatz ein 30stelliges Feld mit int-Werten. Wenn einzelne Werte mit Werten <>0 belegt sind, so können diese ausgewertet werden.

Wenn es sich um Hinweise zum GroupHeader der XML-Datei handelt, so sind diese Hinweise in jedem Datensatz verfügbar (den GroupHeader gibt es nur einmal).

Wenn es sich um Hinweise für die Auftraggeberdaten handelt, so sind diese Hinweise in jedem Datensatz des Sammlers enthalten.

Bei einem gültigen SEPA-XML Datensatz sollten alle Hinweiswerte mit dem Wert 0 belegt sein.

Die Hinweis-Kennziffern haben folgende Bedeutung:

| | |
|---|---|
| -301 | Es wurde kein Name für den Einreicher der Zahlungen gefunden. |
| -302 | Es wurden Adressdaten im GroupHeader angeliefert, jedoch kein Länderkennzeichen übergeben. |
| -303 | Es wurden keine Daten innerhalb des GropHeaders gefunden. |
| -304 | Die XML-Datei enthält keinen GroupHeader. |
| -401 | In den Sammlerdaten wurde kein Name für den Auftraggeber der Zahlung gefunden. |
| -402 | Es wurden Adressdaten für den Auftraggeber angeliefert, jedoch kein Länderkennzeichen übergeben. |
| -403 | Es wurden keine Auftraggeberdaten gefunden. |
| -404 | Für den Auftraggeber des Sammlers wurde kein BIC gefunden. |
| -501 | Es wurden keine Empfängerdaten (Zahlungsaufträge) gefunden. |
| -502 | Es wurde keine EndToEnd-Id für den Zahlungsauftrag gefunden. |
| -503 | Es wurde kein Betrag für den Zahlungsauftrag gefunden. |
| -504 | Es wurde kein Name für den Empfänger des Zahlungsauftrags gefunden. |
| -505 | Es wurden Adressdaten für den Zahlungsempfänger angeliefert, jedoch kein Länderkennzeichen übergeben. |
| -506 | Für den Zahlungsempfänger wurde keine IBAN angegeben. |

# 46 SepaTools_XMLLesenClose

## *46.1 Zweck der Funktion*

Diese Funktion schließt die temporäre Datei und löscht diese.

Wurde die Funktion SepaTools_XMLLesen mit dem Returnwert -3 (Ende der Datei) beendet, so braucht diese Funktion nicht aufgerufen werden. Ein mehrfacher Aufruf ist unschädlich.

## *46.2 Funktionsaufruf*

**void SepaTools_XMLLesenClose()**

## *46.3 Parameter*

keine.

## *46.4 Returncodes*

keine.

# 47 SepaTools_GetBIC

## *47.1 Zweck der Funktion*

Über diese Funktion kann aufgrund einer deutschen Bankleitzahl der dazugehörende BIC ermittelt werden.

**Hinweis:**

**Der zurückgelieferte BIC entspricht dem bei der Deutschen Bundesbank hinterlegten BIC.**

**Bei der Umsetzung von Bankleitzahl und Kontonummer in BIC und IBAN über die Funktion SepaTools_ConvertBLZKonto werden auch die von den Kreditinstituten gemeldeten Sonderfaktoren berücksichtig. Der über diese Funktion zurückgelieferte BIC kann daher abweichen und ist für den SEPA-Zahlungsverkehr zu verwenden.**

## *47.2 Funktionsaufruf*

**int SepaTools_GetBIC(const char *BLZ, char *BIC)**

## *47.3 Parameter*

| | |
|---|---|
| ***BLZ*** | Übergeben Sie hier einen Pointer auf die Bankleitzahl, für die der BIC ermittelt werden soll. Die Bankleitzahl muss nullterminiert 8stellig (für Deutschland) oder 5stellig (für Österreich) übergeben werden. |
| ***BIC*** | Übergeben Sie hier einen Pointer auf einen Speicherbereich, in dem der zu ermittelnde BIC gespeichert werden kann. Der Speicherbereich muss mindestens 11+1 Zeichen aufnehmen können. |

## *47.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Der BIC wurde ermittelt. |
| -1 | Der BIC konnte nicht ermittelt werden, weil die Bankleitzahl nicht existiert. |
| -2 | Die Bankleitzahl wurde gefunden. Für diese Bankleitzahl ist aber kein BIC hinterlegt. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 48 SepaTools_CheckIBAN

## *48.1 Zweck der Funktion*

Über diese Funktion kann eine IBAN auf ihre Richtigkeit geprüft werden. Ungültige zeichen in der IBAN werden vorher entfernt.

## *48.2 Funktionsaufruf*

**int SepaTools_CheckIBAN(const char *IBAN)**

## *48.3 Parameter*

| | |
|---|---|
| ***IBAN*** | Übergeben Sie hier einen Pointer auf die IBAN, die geprüft werden soll. |

## *48.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die IBAN ist gültig. |
| -1 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. |
| -2 | Es wurde keine IBAN übergeben (Leerstring). |
| -3 | Die Länderkennung der IBAN bezieht sich auf ein Land, das keine IBAN hat. Die IBAN kann damit nicht verwendet werden. |
| -4 | Die Länge der IBAN entspricht nicht der für das angegebene Land erforderlichen Länge. |
| -5 | Die Prüfzahl der IBAN (Stelle 3 und 4) darf nur aus Ziffern bestehen. |
| -6 | Die Prüfzahl der IBAN ist ungültig. Die IBAN kann für SEPA nicht verwendet werden. |
| -8 | Dieser Fehlercode kann nur bei deutschen IBANs auftreten.<br><br>Die IBAN ist formal korrekt. Allerdings ist die Kontonummer, die für die IBAN-Ermittlung verwendt wurde, nicht korrekt. Die Prüfzffer ist ungültig. |
| -12 | Dieser Fehlercode kann nur bei deutschen IBANs auftreten.<br><br>Die IBAN ist formal korrekt. Allerdings wurde die Bankleitzahl, die für die IBAN-Ermittlung verwendt wurde, nicht im BLZ-Verzeichnis der Deutschen Bundesbank gefunden. |
| -13 | Die Kontonummer innerhalb der IBAN ist ungültig. Falsche Prüfziffer. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 49 SepaTools_CheckIBAN_Strong

## *49.1 Zweck der Funktion*

Über diese Funktion kann eine IBAN auf ihre Richtigkeit geprüft werden. Im Gegensatz zur Funktion SepaTools_CheckIBAN, werden hier keine ungültigen Zeichen aus der IBAN vor der Berechnung entfernt.

Bei ungülten Zeichen (auch bei Kleinbuchstaben) liefert die Funktion einen negativen Fehlercode.

## *49.2 Funktionsaufruf*

**int SepaTools_CheckIBAN_Strong(const char *IBAN)**

## *49.3 Parameter*

| | |
|---|---|
| ***IBAN*** | Übergeben Sie hier einen Pointer auf die IBAN, die geprüft werden soll. |

## *49.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die IBAN ist gültig. |
| -1 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. |
| -2 | Es wurde keine IBAN übergeben (Leerstring). |
| -3 | Die Länderkennung der IBAN bezieht sich auf ein Land, das keine IBAN hat. Die IBAN kann damit nicht verwendet werden. |
| -4 | Die Länge der IBAN entspricht nicht der für das angegebene Land erforderlichen Länge. |
| -5 | Die Prüfzahl der IBAN (Stelle 3 und 4) darf nur aus Ziffern bestehen. |
| -6 | Die Prüfzahl der IBAN ist ungültig. Die IBAN kann für SEPA nicht verwendet werden. |
| -7 | Die IBAN enthält ungültige Zeichen. |
| -8 | Dieser Fehlercode kann nur bei deutschen IBANs auftreten.<br><br>Die IBAN ist formal korrekt. Allerdings wurde die Bankleitzahl, die für die IBAN-Ermittlung verwendt wurde, nicht im BLZ-Verzeichnis der Deutschen Bundesbank gefunden. |
| -9 | Dieser Fehlercode kann nur bei deutschen IBANs auftreten.<br><br>Die IBAN ist formal korrekt. Allerdings ist die Kontonummer, die für die IBAN-Ermittlung verwendt wurde, nicht korrekt. Die Prüfzffer ist ungültig. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 50 SepaTools_GetBLZKonto

## *50.1 Zweck der Funktion*

Diese Funktion ermittelt aus einer übergebenen IBAN die Bankleitzahl und die Kontonummer, sowie den BIC und den Namen der Bank. Die Funktion kann für Deutsche, Österreichische und Niederländische IBANs angewendet werden.

## *50.2 Funktionsaufruf*

**int SepaTools_GetBLZKonto(const char *IBAN, BLZKontoStruct *BLZKonto)**

## *50.3 Parameter*

***IBAN*** Übergeben Sie hier einen Pointer auf die IBAN aus der die Daten gewonnen werden sollen.

***BLZKonto*** In dieser Struktur werden die ermittelten Daten zurückgegeben. Die Struktur ist im Folgenden dargestellt.

struct BLZKontoStruct

| | | | |
|---|---|---|---|
| { | char | Land[2+1] | Länderkennung DE, AT, NL, PL, CH oder LI |
| | char | BLZ[8+1] | Die zurückgelieferte Bankleitzahl |
| | char | Konto[11+1] | Die zurückgelieferte Kontonummer |
| | char | BIC[11+1] | Der ermittelte BIC |
| | char | BankName[50+1] | Der aus dem SCL-Directory ermittelte Name der Bank |
| | char | Ort[30+1] | Der aus dem BLZ-Verzeichnis |
| | char | KontoLang[20+1] | Lange Kontonummer, z.B. für Schweiz und Liechtenstein [1)] |
| | char | Reserve [148] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen

1) In der Schweiz und in Liechtentein können die Kontonummern länger als 11 Zeichen sein. Um die Struktur (aus Sicht der Bestandsanwender) nicht zu verändern, wurde dieseszusätzliche Feld im Reservebereich untergebracht.

   Im Feld „Konto“ steht dann die u.U. auf 11 Stellen gekürzte Kontonummer. ImFeld „KontoLang“ steht dann die vollständige Kontonummer.

### *50.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die IBAN ist gültig. Die Umsetzung ist erfolgt. |

**Achtung:**

Die folgenden drei positiven Rückgabecodes sind Bit-Codiert und werden als ein Wert zurückgegeben. Die Bit-Werte sind in rot dargestellt. Diese Bit codierten Werte dienen nur der Informtion und treten nur auf, wenn der Vorgang erfolgreich abgeschlossen werden konnte.

| | |
|---|---|
| **1** | Alles verlief erfolgreich. Die IBAN ist gültig. Die Umsetzung ist erfolgt. Es konnte ein BIC ermittelt werden. Dieser BIC ist aber über die EBA nicht erreichbar. |
| **2** | Die übergebene IBAN enthält Kleinbuchstaben. Dies ist nicht erlaubt. Die Kleinbuchstaben wurden in Großbuchstaben umgesetzt. |
| **4** | In der übergebenen IBAN waren ungültige Zeichen enthalten. Die ungültigen Zeichen wurden entfernt. |
| -1 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. |
| -2 | Es wurde keine IBAN übergeben (Leerstring). |
| -3 | Die Länderkennung der IBAN bezieht sich auf ein Land, das keine IBAN hat. Die IBAN kann damit nicht verwendet werden. |
| -4 | Die Länge der IBAN entspricht nicht der für das angegebene Land erforderlichen Länge. |
| -5 | Die Prüfzahl der IBAN (Stelle 3 und 4) darf nur aus Ziffern bestehen. |
| -6 | Die Prüfzahl der IBAN ist ungültig. Die IBAN kann für SEPA nicht verwendet werden. |
| -7 | Die Länderkennung der übergebenen IBAN lautet weder auf „DE“, „AT“, „NL“, „PL“, „CH“ noch auf „LI“. |
| -8 | Es konnte kein BIC ermittelt werden. Wahrscheinlich ist die Bankleitzahl ungültig. Dieser Fehlercode wird aber auch ausgegeben, wenn bei einer deutchen IBAN die Prüfziffer des Kontos innerhalb der IBAN falsch ist. |
| -10 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -11 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -12 | Dieser Fehlercode wird ausgegeben, wenn die BLZ innerhalb der IBAN nicht im deutschen BLZ-Bestand gefunden wurde. |
| -13 | Die Kontonummer (Prüfziffer) innerhalb der deutschen IBAN ist ungültig. |

| | |
|---|---|
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 51 SepaTools_CheckIBANLand

## *51.1 Zweck der Funktion*

Über diese Funktion kann festgestellt werden, ob ein Land eine IBAN hat und/oder ob das Land an SEPA teilnimmt.

## *51.2 Funktionsaufruf*

**int SepaTools_CheckIBANLand(const char *Land, const int SEPA)**

## *51.3 Parameter*

***Land*** Übergeben Sie bitte hier einen Pointer auf den 2stelligen ISO-Code (z.B. DE oder IT) des Landes, das Sie prüfen wollen.

***SEPA*** Hier können Sie angeben, ob die Prüfung auf das Land sich auf alle Länder bezieht, die eine IBAN haben oder nur auf Länder mit einer IBAN, die auch am SEPA-Zahlungsverkehr teilnehmen.

Folgende Werte können übergeben werden:

0 Die Suche erfolgt in der Tabelle aller Länder, die eine IBAN haben unabhängig davon, ob sie auch an SEPA teilnehmen.

1 (≠ 0) Die Suche erfolgt in der Tabelle nur nach den Ländern, die eine IBAN haben und auch am SEPA-Zahlungsverkehr teilnehmen.

## *51.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die IBAN ist gültig. |
| -1 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. |
| -2 | Es wurde kein Land übergeben (Leerstring). |
| -3 | Die Länderkennung wurde in den Tabellen nicht gefunden. Dies bezieht sich entweder auf alle Länder, die eine IBAN haben (Parameter SEPA = 0) oder nur auf die Länder, die eine IBAN haben und auch am SEPA-Zahlungsverkehr teilnehmen (Parameter SEPA = 1). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 52 SepaTools_CheckCI

## *52.1 Zweck der Funktion*

Über diese Funktion kann eine Gläubiger-Identifikations-Nummer (CI) auf ihre Richtigkeit geprüft werden.

## *52.2 Funktionsaufruf*

**int SepaTools_CheckCI(const char *CI)**

## *52.3 Parameter*

| | |
|---|---|
| ***CI*** | Übergeben Sie hier einen Pointer auf die CI, die geprüft werden soll. |

## *52.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die CI ist gültig. |
| -1 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. |
| -2 | Es wurde keine CI übergeben (Leerstring). |
| -3 | Die Länderkennung der CI bezieht sich auf ein Land, das nicht am SEPA-Verfahren teilnimmt. Die CI kann für SEPA nicht verwendet werden. |
| -4 | Die Prüfzahl der CI (Stelle 3 und 4) darf nur aus Ziffern bestehen. |
| -5 | Die Prüfzahl der CI ist ungültig. Die CI kann für SEPA nicht verwendet werden. |
| -6 | Die Länge der übergebenen CI ist ungültig. Diese Prüfung erfolgt, wenn die erforderlichen Informationen vorliegen. Die Informationen berufen auf das vom EPC zur Verfügung gestellten Verzeichnis.<br><br>Es können nicht alle CI's vernünftig geprüft werden, da nicht alle Informationen vorliegen und es Länder gibt, die CI's mit variabler Längen haben. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

## *52.5 Bekannte Längen von CIs*

Bei folgenden Ländern wird die Länge der CI aufgrund der Informationen vom EPC geprüft.

| *Kurz* | *Land* | *Länge CI* |
|---|---|---|
| AT | Österreich | 18 |
| BE | Belgien | 17 oder 20 |
| CH | Schweiz | 18 |
| CY | Zypern | 11 |
| CZ | Tschechien | 12 |

| Kurz | Land | Länge CI |
|---|---|---|
| DE | Deutschland | 18 |
| DK | Dänemark | 15 |
| EE | Estland | 20 |
| ES | Spanien | 16 |
| FI | Finnland | 11 |
| FR | Frankreich | 13 |
| GR | Griechenland | 12 |
| HR | Kroatien | 18 |
| HU | Ungarn | 16 |
| IE | Irland | 13 |
| IT | Italien | 23 |
| LI | Liechtenstein | 18 |
| LU | Luxemburg | 26 |
| LT | Litauen | 16 |
| LV | Lettland | 18 |
| MC | Monaco | 13 |
| MT | Malta | 17 |
| NL | Niederlande | 19 |
| NO | Norwegen | 16 |
| PT | Portugal | 13 |
| SE | Schweden | 17 |
| SI | Slowenien | 15 |
| SK | Slowakei | 18 |
| SM | San Marino | 23 |

# 53 SepaTools_CheckESR

## *53.1 Zweck der Funktion*

Über diese Funktion kann eine ESR Referenznummer, wie teilweise in der Schweiz verwendet auf ihre Richtigkeit geprüft werden. Geprüft können nur aktuelle ESR-Nummern mit einer Länge von 27 Stellen werden.

Ältere Versionen der ESR-Numer werden nicht unterstützt.

## *53.2 Funktionsaufruf*

**int SepaTools_CheckESR(const char *ESR)**

## *53.3 Parameter*

| | |
|---|---|
| ***ESR*** | Übergeben Sie hier einen Pointer auf die ESR-Nummer, die geprüft werden soll. |

## *53.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die ESR ist gültig. |
| -1 | Die ESR-Numer (Referenz) ist ungültig. |
| -2 | Die Länge der ESR-Nummer ist ungültig (ungleich 27). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 54 SepaTools_GetESRPrZiff

## *54.1 Zweck der Funktion*

Über diese Funktion wird die Prüffziffer einer ESR-Nummer ermittelt. Wird die ESR-Nummer 26stellig übergeben, so wird die korrekte Prüfziffer angehängt und die ESR-Nummer wird 27stellig zurückgegeben.

Die Prüfziffer kann nur für aktuelle ESR-Nummern mit einer Länge von 27 Stellen (incl. Prüfziffer) ermittelt werden. Ältere Versionen der ESR-Nummer werden nicht unterstützt.

## *54.2 Funktionsaufruf*

**int SepaTools_GetESRPrZiff(char *ESR)**

## *54.3 Parameter*

| | |
|---|---|
| ***ESR*** | Übergeben Sie hier einen Pointer auf die ESR-Nummer, für die die Prüfziffer errechnet werden soll. In diesem Pointer wird auch das Ergebnis zurückgegeben. |

## *54.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die ESR ist gültig. |
| -2 | Die Länge der ESR-Nummer ist ungültig (ungleich 27 und ungleich 26). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 55 SepaTools_CheckSEPATeilnahme

## *55.1 Zweck der Funktion*

Über diese Funktion kann ermittelt werden, an welchen SEPA-Verfahren eine Bank teilnimmt.

## *55.2 Funktionsaufruf*

**int SepaTools_CheckSEPATeilnahme(const char *BIC, int *Info)**

## *55.3 Parameter*

***BIC*** Übergeben Sie hier einen Pointer auf den BIC, für den die Verfahren ermittelt werden sollen. Der BIC ist entweder 8stellig oder 11stellig.

***Info*** Übergeben Sie hier einen Pointer auf einen Integerwert, in dem die Informationen gespeichert werden sollen. Die Informationen sind bitweise (Oder-Verknüpfung) abgelegt und bedeuten Folgendes:

1 Die Bank nimmt am Überweisungsverfahren (CT) teil.

2 Die Bank nimmt am normalen Lastschriftverfahren (DD) teil (Core-Lastschrift.

4 Die Bank nimmt an der B2B-Lastschrift teil.

8 Die Bank nimmt an der COR1-Lastschrift teil (kürzere Einreichungsfristen). Bitte beachten Sie, dass COR1-Lastschriften in der Regel nur im Inland möglich sind.

16 Reserviert.

32 Die Bank nimmt am Instant Verfahren (SCT Inst) teil.

Die Werte werden bitweise verknüpft. Wenn eine Bank an allen vier Verfahren teilnimmt, so würde hier der Wert 15 zurückgegeben. Nimmt die Bank nur an den ersten beiden Verfahren teil, so wird der Wert 3 zurückgeliefert.

## *55.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| -1 | Für den angegebenen BIC wurden keine Daten gefunden. |
| -2 | Es wurde ein BIC mit ungültiger Länge (<>8 und <>11) übergeben. |
| -3 | Der übergebene BIC existiert, nimmt aber nicht an SEPA teil. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 56 SepaTools_ConvertBLZKonto

## *56.1 Zweck der Funktion*

Über diese Funktion kann aus einer Bankleitzahl und der Kontonummer eine IBAN und ein BIC ermittelt werden. Da nicht alle Kreditinstitute ihre Berechnungsregeln offen gelegt haben, kann es hier auch zu Fehlern kommen.

Wurden spezielle BICs und IBANs mit den dazugehörigen Bankleitzahlen und Kontonummern hinterlegt, so wird diese Übersetzungstabelle in die Ermittlung von BIC und IBAN aus Bankleitzahl und Kontonummer mit einbezogen.

Diese Funktion kann in weiten Teilen auch mit österreichischen Bankverbindungen umgehen. Aufgrund der übergebenen Bankleitzahl (5stellig für Österreich und 8stellig für Deutschland) entscheidet die API, ob der Bankleitzahlenbestand der Deutschen Bundesbank oder der Österreichischen Nationalbank verwendet wird.

**Hinweis:**

**Bei österreichischen Banken erfolgt keine Prüfung der Prüfziffer.**

**Bei der Umsetzung von Bankleitzahl und Kontonummer in BIC und IBAN über die Funktion SepaTools_ConvertBLZKonto werden auch die von den Kreditinstituten gemeldeten Sonderfaktoren berücksichtig. Der über diese Funktion zurückgelieferte BIC kann daher von dem über die Funktion SepaTools_GetBIC zurückgelieferten BIC abweichen und ist für den SEPA-Zahlungsverkehr zu verwenden.**

## *56.2 Funktionsaufruf*

**int SepaTools_ConvertBLZKonto(XMLConvertStruct *ConvertStruct)**

## *56.3 Parameter*

***ConvertStruct*** In dieser Struktur werden die Werte übergeben und die Ergebnisse zurückgeliefert.

In einigen Fällen kann es vorkommen, dass im Rahmen der Umrechnung eine BLZ ausgetauscht, oder eine Kontonummer ergänzt wird. In diesen Fällen werden dann auch die modifizierten Ausgangswerte zurückgeliefert.

```
struct    XMLConvertStruct

{   char    BLZ[8+1]          Bankleitzahl, die verwendet werden soll.
    char    Konto[11+1]       Kontonummer, die verwendet werden soll.
    char    BIC[11+1]         Der ermittelte BIC wird zurückgeliefert.
    char    IBAN[35+1]        Die ermittelte IBAN wird zurückgeliefert.
}
```

## *56.4 Returncodes*

**Hinweis:**

Bei Returncodes >=0 (keine Fehler), werden im Returncode auch zusätzliche Informationen gespeichert. **Diese Informationen sind bitweise abgelegt**.

**Die Bitbelegungen für den Returncode >= 0 haben folgende Bedeutung:**

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| 1 | Alles verlief erfolgreich. Aufgrund von Sonderregelungen einiger Kreditinstitute wurde die Kontonummer modifiziert. Die modifizierte Kontonummer wird in der Struktur zurückgegeben. |
| 2 | Alles verlief erfolgreich. Aufgrund von Sonderregelungen einiger Kreditinstitute wurde die Bankleitzahl durch eine andere Bankleitzahl ersetzt. Die neue Bankleitzahl wird in der Struktur zurückgegeben. |
| 4 | Die Bankleitzahl ist zur Löschung vorgemerkt und es wurde keine Nachfolge-Bankleitzahl angegeben. Da nicht bekannt ist, wann diese Bankleitzahl tatsächlich gelöscht wird, kann die Bankleitzahl bis auf weiteres verwendet werden.<br><br>Es ist aber ein Hinweis darauf, dass diese Bankleitzahl irgendwann gelöscht wird. |
| 8 | Die Bankleitzahl ist zur Löschung vorgemerkt. Es wurde aber eine Nachfolge-Bankleitzahl angegeben, die zur Ermittlung der IBAN herangezogen wurde. Die Nachfolge-Bankleitzahl kann verwendet werden. |
| 16 | Die Umwandlung von BLZ und Kontonummer in eine IBAN wurde durchgeführt. Hierzu wurden aber Erfahrungswerte verwendet. Die Wahrscheinlichkeit einer korrekten Umwandlung ist zwar hoch aber nicht garantiert.<br><br>Bitte überprüfen Sie das Ergebnis. |
| 32 | Die errechnete IBAN kann verwendet werden. Sie ist aber nicht eindeutig. Eine Rückfrage beim Kunden wird empfohlen. |

**Die Returncodes ab hier sind wie bisher als Integer-Werte auszuwerten.**

| | |
|---|---|
| -2 | Die übergebene Bankleitzahl hat eine Länge von ungleich 5 oder 8. |
| -3 | Die übergebene Kontonummer ist nur einstellig. Die Kontonummer muss einen Wert von größer als 9 haben. |
| -4 | Die übergebene Bankleitzahl wurde nicht im Bankleitzahlenbestand der Deutschen Bundesbank, bzw. der Österreichischen Nationalbank gefunden. |

| | |
|---|---|
| -5 | Die übergebene Bankleitzahl wird von der Deutschen Bundesbank gelöscht und kann nicht verwendet werden (nicht bei österreichischen Bankleitzahlen). In der nächsten Ausgabe des Bankleitzahlenbestandes ist diese Bankleitzahl nicht mehr enthalten. |
| -6 | Die Prüfziffer der Kontonummer ist falsch. Die Kontonummer kann zur Umwandlung nicht verwendet werden (nicht bei österreichischen Bankleitzahlen). |
| -7 | Die Berechnung der IBAN ist nicht eindeutig. die IBAN kann daher aus Sicherheitsgründen nicht verwendet werden. |
| -8 | Für die übergebene Bankleitzahl wurde kein BIC gefunden. |
| -9 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -10 | Die Kontoführende Bank hat diese Bankleitzahl in Verbindung mit der Kontonummer von der IBAN-Berechnung ausgenommen. Eine IBAN-Berechnung kann nicht erfolgen. |
| -11 | Die übergebene Bankleitzahl enthält ungültige Zeichen oder ist leer. |
| -12 | Die übergebene Kontonummer enthält ungültige Zeichen oder ist leer. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 57 SepaTools_SetErfahrung

## *57.1 Zweck der Funktion*

Diese Funktion wird nicht mehr benötigt. Die Informationen der betroffenen Banken haben einen Stand erreicht, der mit Erfahrungswerten derzeit nicht mehr verbessert werden kann

Aus Kompatibilitätsgründen ist die Funktion aber weiterhin implementiert. Der Aufruf dieser Funktion hat aber keine Auswirkung mehr.

## *57.2 Funktionsaufruf*

**int SepaTools_SetErfahrung(int Flag)**

## *57.3 Parameter*

| | |
|---|---|
| ***Flag*** | Übergeben Sie bitte hier einen Integer-Wert als Dummy. |

## *57.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Es wird immer der Wert 0 zurückgegeben. |

# 58 SepaTools_GetBankInfo

## *58.1 Zweck der Funktion*

Über diese Funktion können Informationen über die an SEPA teilnehmenden Banken abgerufen werden.

## *58.2 Funktionsaufruf*

**int SepaTools_GetBankInfo(const char *BIC, XMLBankInfoStruct *BankInfoStruct)**

## *58.3 Parameter*

***BIC*** Geben Sie hier bitte den BIC an, für den die Informationen abgerufen werden sollen. Der BIC hat eine Länge von 8 Stellen oder 11 Stellen.

***BankInfoStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct XMLBankInfoStruct

```
{   int     Art              Verfahren, an denen die Bank teilnimmt 1).
    char    BIC[11+1]        BIC der Bank.
    char    BankName[50+1]   Bezeichnung der Bank.
    char    LandKurz[2+1]    Länderkürzel des Landes der Bank. Z.B. DE oder AT.
    char    Land[20+1]       Bezeichnung des Landes.
    char    Strasse[27+1]    Anschrift der Bank (häufig nicht verfügbar).
    char    Ort[50+1]        Ort der Bank (häufig nicht verfügbar).
}
```

Zusätzliche Erklärungen

1) Übergeben wird hier ein Integerwert, in dem die Informationen gespeichert werden sollen. Die Informationen sind bitweise (Oder-Verknüpfung) abgelegt und bedeuten Folgendes:

   1 Die Bank nimmt am Überweisungsverfahren (CT) teil.

   2 Die Bank nimmt am normalen Lastschriftverfahren (DD) teil (Core-Lastschrift).

   4 Die Bank nimmt an der B2B-Lastschrift teil.

   8 Die Bank nimmt an der COR1-Lastschrift teil (kürzere Einreichungsfristen). Bitte beachten Sie, dass COR1-Lastschriften in der Regel nur im Inland möglich sind.

   16 Reserviert.

   32 Die Bank nimmt am Instant Verfahren (SCT Inst) teil.

   Die Werte werden bitweise verknüpft. Wenn eine Bank an den ersten vier Verfahren teilnimmt, so würde hier der Wert 15 zurückgegeben. Nimmt die Bank nur an den ersten beiden Verfahren teil, so wird der Wert 3 zurückgeliefert.

### *58.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| 1 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. Der BIC wurde aber angepasst (Es wurde XXX angehängt). |
| -1 | Es konnten keine Informationen für den übergebenen BIC ermittelt werden. Möglicherweise ist der BIC falsch, oder die Bank nimmt an keinem der SEPA-Verfahren teil. |
| -2 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -3 | Der übergebene BIC existiert, nimmt aber nicht an SEPA teil. |
| -4 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 59 SepaTools_GetBICInfo

## *59.1 Zweck der Funktion*

Über diese Funktion können Informationen aus dem BLZ-Bestand der Deutschen Bundesbank oder der Österreichischen Nationalbank abgerufen werden. Als Schlüsselbegriff für den Abruf der Informationen dient hier der BIC.

## *59.2 Funktionsaufruf*

**int SepaTools_GetBICInfo(const char *BIC, XMLBLZStruct *BLZStruct)**

## *59.3 Parameter*

***BIC*** Geben Sie hier als Konstante den BIC als Zeichenkette, nach dem die Informationen gesucht werden sollen.

***BLZStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct XMLBLZStruct

| | | | |
|---|---|---|---|
| { | char | BLZ[8+1] | Bankleitzahl |
| | char | NameKurz[27+1] | Kurzname der Bank |
| | char | NameLang[35+1] | Langname der Bank |
| | char | PLZ[5+1] | Postleitzahl |
| | char | Ort[25+1] | Ort der Bank |
| | int | EigeneBLZ | Wert=1, wenn BLZ-führende Stelle, Wert=2 wenn Filiale |
| | char | Verfahren[2+1] | Prüfziffernverfahren lt. Deutscher Bundesbank |
| | int | DelHinweis | Wert=1, wenn BLZ zur Löschung vorgemerkt ist, sonst 0 |
| | char | ChangeKz | Änderungskennzeichen:<br>A = Neuer Datensatz<br>D = Löschung<br>U = Unverändert<br>M = Modifiziert |
| | char | NachfolgeBLZ[8+1] | Nachfolge-BLZ, wenn vorhanden |
| | char | BIC[11+1] | Zur BLZ gehörender BIC [1)] |
| } | | | |

Zusätzliche Erklärungen

1) Bei dem hier zurückgelieferten BIC handelt es sich um den BIC der im Bankleitzahlenbestand der Deutschen Bundesbank hinterlegt ist. Der erforderliche BIC für den SEPA-Zahlungsverkehr kann von diesem BIC abweichen, da er auch noch von anderen Faktoren beeinflusst wird.

   Verwenden Sie für den SEPA-Zahlungsverkehr daher immer den BIC, der mit der Funktion SepaTools_ConvertBLZKonto ermittelt wurde (Bei einigen Banken ist der BIC auch abhängig von einem bestimmten Kontonummernkreis).

### *59.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| -1 | Für den übergebenen BIC wurden keine Informationen gefunden. |
| -2 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -3 | Der übergebene BIC gehört weder zu Deutschland (DE), noch zu Österreich (AT). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 60 SepaTools_GetIBANLand

## *60.1 Zweck der Funktion*

Über diese Funktion können Informationen die aktuellen Länder mit einer IBAN abgerufen werden. Die Funktion muss dazu mehrfach aufgerufen werden, bis der Rückgabewert <>0 ist.

## *60.2 Funktionsaufruf*

**int SepaTools_GetIBANLand(int First, XMLIBANLandStruct *IBANLandStruct)**

## *60.3 Parameter*

***First*** Geben Sie hier einen Wert > 0 an, wenn Sie die Funktion zum ersten Mal aufrufen. Damit wird die Funktion intern initialisiert. Bei anderen Funktionsaufrufen geben Sie hier bitte den Wert 0 an.

***IBANLandStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct XMLIBANLandStruct

```
{   char    LandKurz[2+1]       Die Kurzbezeichnung des Landes, z.B. DE oder AT.
    char    LandName[30+1]      Der Name des Landes.
    char    IntLandName[20+1]   Der internationale Name des Landes, z.B. AUSTRIA.
    int     LangeIBAN           die Länge der IBAN dieses Landes (Deutschland = 22).
    char    IBAN[35+1]          Eine beispielhafte IBAN für dieses Land.
    int     IsSepa              Wer=1, wenn das Land auch an SEPA teilnimmt, sonst 0.
}
```

## *60.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| -1 | Sie sind am Ende der Datenbank angekommen. Es stehen keine weiteren Daten mehr zur Verfügung. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 61 SepaTools_GetSEPABank

## *61.1 Zweck der Funktion*

Über diese Funktion können Informationen die an SEPA teilnehmenden Banken abgerufen werden. Die Funktion muss dazu mehrfach aufgerufen werden, bis der Rückgabewert <>0 ist.

**Hinweis:**

**Bitte beachten Sie, dass es sich hier um sehr große Datenvolumen handelt. Es werden sehr viele Datensätze zurückgegeben.**

## *61.2 Funktionsaufruf*

**int SepaTools_GetSEPABank (int First, char *Land, char *Ort,**
**XMLBankInfoStruct *BankInfoStruct)**

## *61.3 Parameter*

***First*** Geben Sie hier einen Wert > 0 an, wenn Sie die Funktion zum ersten Mal aufrufen. Damit wird die Funktion intern initialisiert. Bei den weiteren Funktionsaufrufen geben Sie hier bitte den Wert 0 an.

***Land*** Geben Sie bitte hier die Länder-Kurzbezeichnung (z.B. DE, AT usw.) für das Land ein, für das Sie an SEPA teilnehmende Banken suchen wollen.

Wenn Sie hier einen Leerstring übergeben, so werden alle Länder nach an SEPA teilnehmenden Banken durchsucht.

***Ort*** Geben Sie hier als Suchbegriff den Namen eines Ortes oder den Teilnahmen eines Ortes (z.B. Frankf) ein. Wenn Sie auch eine Länder-Kurzbezeichnung angegeben haben, so werden Banken in allen Orten des angegebenen Landes gesucht.

Haben Sie keine Länder-Kurzbezeichnung eingegeben, so werden die Banken entsprechend des Suchcodes (Orte) unabhängig von einem Land gesucht.

Auch die Angabe eines Leerstrings ist möglich. Dann werden alle Banken gesucht.

***BankInfoStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct XMLBankInfoStruct

| | | | |
|---|---|---|---|
| { | int | Art | Verfahren, an denen die Bank teilnimmt [1)]. |
| | char | BIC[11+1] | BIC der Bank. |
| | char | BankName[50+1] | Bezeichnung der Bank. |
| | char | LandKurz[2+1] | Länderkürzel des Landes der Bank. Z.B. DE oder AT. |
| | char | Land[20+1] | Bezeichnung des Landes. |
| | char | Strasse[27+1] | Anschrift der Bank (häufig nicht verfügbar). |
| | char | Ort[50+1] | Ort der Bank (häufig nicht verfügbar). } |

Zusätzliche Erklärungen

1) Übergeben wird hier ein Integerwert, in dem die Informationen gespeichert werden sollen. Die Informationen sind Bitweise (Oder-Verknüpfung) abgelegt und bedeuten Folgendes:

   1 Die Bank nimmt am Überweisungsverfahren (CT) teil.

   2 Die Bank nimmt am normalen Lastschriftverfahren (DD) teil (Core-Lastschrift.

   4 Die Bank nimmt an der B2B-Lastschrift teil.

   8 Die Bank nimmt am COR1 Verfahren teil

   16 Reserviert

   32 Die Bank nimmt am Instant Verfahren (SCT Inst) teil.

   Die Werte werden bitweise verknüpft. Wenn eine Bank an den ersten drei Verfahren teilnimmt, so würde hier der Wert 7 zurückgegeben. Nimmt die Bank nur an den ersten beiden Verfahren teil, so wird der Wert 3 zurückgeliefert.

### *61.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| -1 | Sie sind am Ende der Datenbank angekommen. Es stehen keine weiteren Daten mehr zur Verfügung. |
| -2 | Die Datenbank Sepa.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -3 | Als Länderkennzeichen wurde ein einstelliger Wert übergeben. Es darf nur ein zweistelliger Wert (Z.B. DE oder AT usw.) oder ein Leerstring übergeben werden. |
| -4 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 62 SepaTools_GetBLZ

## *62.1 Zweck der Funktion*

Über diese Funktion können Informationen aus dem BLZ-Bestand der Deutschen Bundesbank oder der Österreichischen Nationalbank abgerufen werden. Die Funktion muss ist dazu mehrfach aufgerufen werden, bis der Rückgabewert <>0 ist.

## *62.2 Funktionsaufruf*

**int SepaTools_GetBLZ(char *Land, int *First, char *Ort, XMLBLZStruct *BLZStruct)**

## *62.3 Parameter*

***Land*** Wenn Sie als Länderkennzeichen der Wert AT für Österreich angeben, so wird im Bankleitzahlenbestand der österreichischen Nationalbank gesucht.

In allen anderen Fällen, auch bei der Übergabe eines Leerstrings, wird im Bankleitzahlenbestand der Deutschen Bundesbank gesucht.

***First*** Geben Sie hier einen Wert > 0 an, wenn Sie die Funktion zum ersten Mal aufrufen. Damit wird die Funktion intern initialisiert. Nach dem ersten Aufruf dieser Funktion wird der Wert von der API automatisch auf 0 gesetzt.

***Ort*** Geben Sie hier als Suchbegriff den Namen eines Ortes oder den Teilnahmen eines Ortes (z.B. Frankf) ein.

Auch die Angabe eines Leerstrings ist möglich. Dann werden alle Bankleitzahlen gesucht.

***BLZStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct XMLBLZStruct

| | | | |
|---|---|---|---|
| { | char | BLZ[8+1] | Bankleitzahl |
| | char | NameKurz[27+1] | Kurzname der Bank |
| | char | NameLang[35+1] | Langname der Bank |
| | char | PLZ[5+1] | Postleitzahl |
| | char | Ort[25+1] | Ort der Bank |
| | int | EigeneBLZ | Wert=1, wenn BLZ-führende Stelle, Wert=2 wenn Filiale |
| | char | Verfahren[2+1] | Prüfziffernverfahren lt. Deutscher Bundesbank |
| | int | DelHinweis | Wert=1, wenn BLZ zur Löschung vorgemerkt ist, sonst 0 |
| | char | ChangeKz | Änderungskennzeichen:<br>A = Neuer Datensatz<br>D = Löschung<br>U = Unverändert<br>M = Modifiziert |
| | char | NachfolgeBLZ[8+1] | Nachfolge-BLZ, wenn vorhanden |
| | char | BIC[11+1] | Zur BLZ gehörender BIC [1)] |
| } | | | |

Zusätzliche Erklärungen

1) Bei dem hier zurückgelieferten BIC handelt es sich um den BIC der im Bankleitzahlenbestand der Deutschen Bundesbank hinterlegt ist. Der erforderliche BIC für den SEPA-Zahlungsverkehr kann von diesem BIC abweichen, da er auch noch von anderen Faktoren beeinflusst wird.

   Verwenden Sie für den SEPA-Zahlungsverkehr daher immer den BIC, der mit der Funktion SepaTools_ConvertBLZKonto ermittelt wurde (Bei einigen Banken ist der BIC auch abhängig von einem bestimmten Kontonummernkreis).

## *62.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| -1 | Sie sind am Ende der Datenbank angekommen. Es stehen keine weiteren Daten mehr zur Verfügung. |
| -2 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 63 SepaTools_FindBLZ

## *63.1 Zweck der Funktion*

Über diese Funktion können Informationen aus dem BLZ-Bestand der Deutschen Bundesbank oder der Österreichischen Nationalbank, bezogen auf eine übergebene Bankleitzahl abgerufen werden. Die Funktion muss ist dazu mehrfach aufgerufen werden, bis der Rückgabewert <>0 ist.

Die zurückgelieferten Informationen sind in der Struktur XMLBLZStruct enthalten.

## *63.2 Funktionsaufruf*

**int SepaTools_FindBLZ(char *Land, const char *BLZ, int *First, XMLBLZStruct *BLZStruct)**

## *63.3 Parameter*

***Land*** Wenn Sie als Länderkennzeichen der Wert AT für Österreich angeben, so wird im Bankleitzahlenbestand der österreichischen Nationalbank gesucht.

In allen anderen Fällen, auch bei der Übergabe eines Leerstrings, wird im Bankleitzahlenbestand der Deutschen Bundesbank gesucht.

***BLZ*** Übergeben Sie bitte hier einen Zeiger auf die Bankleitzahl, für die Informationen gesucht werden sollen. Da die Bankleitzahl möglicherweise mehrfach im BLZ-Bestand der Deutschen Bundesbank enthalten ist, muss die Funktion mehrfach aufgerufen werden.

***First*** Geben Sie hier einen Wert > 0 an, wenn Sie die Funktion zum ersten Mal aufrufen. Damit wird die Funktion intern initialisiert. Nach dem ersten Aufruf dieser Funktion wird der Wert von der API automatisch auf 0 gesetzt.

***BLZStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct XMLBLZStruct

```
{   char    BLZ[8+1]            Bankleitzahl
    char    NameKurz[27+1]      Kurzname der Bank
    char    NameLang[35+1]      Langname der Bank
    char    PLZ[5+1]            Postleitzahl
    char    Ort[25+1]           Ort der Bank
    int     EigeneBLZ           Wert=1, wenn BLZ-führende Stelle, Wert=2 wenn Filiale
    char    Verfahren[2+1]      Prüfziffernverfahren lt. Deutscher Bundesbank
    int     DelHinweis          Wert=1, wenn BLZ zur Löschung vorgemerkt ist, sonst 0
    char    ChangeKz            Änderungskennzeichen: A=Neuer Datensatz, D=Löschung,
                                U=Unverändert, M=Modifiziert
    char    NachfolgeBLZ[8+1]   Nachfolge-BLZ, wenn vorhanden
    char    BIC[11+1]           Zur BLZ gehörender BIC [1)]
}
```

Zusätzliche Erklärungen

1) Bei dem hier zurückgelieferten BIC handelt es sich um den BIC der im Bankleitzahlenbestand der Deutschen Bundesbank hinterlegt ist. Der erforderliche BIC für den SEPA-Zahlungsverkehr kann von diesem BIC abweichen, da er auch noch von anderen Faktoren beeinflusst wird.

   Verwenden Sie für den SEPA-Zahlungsverkehr daher immer den BIC, der mit der Funktion SepaTools_ConvertBLZKonto ermittelt wurde (Bei einigen Banken ist der BIC auch abhängig von einem bestimmten Kontonummernkreis).

## *63.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| -1 | Sie sind am Ende der Datenbank angekommen. Es stehen keine weiteren Daten mehr zur Verfügung. |
| -2 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 64 SepaTools_SearchBLZ

## *64.1 Zweck der Funktion*

Über diese Funktion können Informationen aus dem BLZ-Bestand der Deutschen Bundesbank oder der Österreichischen Nationalbank, bezogen auf eine übergebene Bankleitzahl abgerufen werden. Die Funktion muss ist dazu mehrfach aufgerufen werden, bis der Rückgabewert <>0 ist.

Die zurückgelieferten Informationen sind in der Struktur XMLBLZStruct enthalten.

Im Gegensatz zur Funktion SepaTools_FindBLZ kann hier auch eine Bankleitzahl mit weniger als 8 Ziffern (bzw. 5 Ziffern bei österreichischen Bankleitzahlen) als Suchbegriff angegeben werden. Die Suche ist dann erfolgreich, wenn die als Bankleitzahl angegebenen Ziffern mit den ersten Ziffern der Bankleitzahl im Bankleitzahlenbestand übereinstimmen.

Wenn Sie als Bankleitzahl einen Leerstring übergeben, so werden alle Bankleitzahlen des entsprechenden Landes (DE oder AT) aufgelistet.

## *64.2 Funktionsaufruf*

**int SepaTools_SearchBLZ(char *Land, char *BLZ, int *First, XMLBLZStruct *BLZStruct)**

## *64.3 Parameter*

***Land*** Wenn Sie als Länderkennzeichen der Wert AT für Österreich angeben, so wird im Bankleitzahlenbestand der österreichischen Nationalbank gesucht.

In allen anderen Fällen, auch bei der Übergabe eines Leerstrings, wird im Bankleitzahlenbestand der Deutschen Bundesbank gesucht.

***BLZ*** Übergeben Sie bitte hier einen Zeiger auf die Bankleitzahl oder den Teil einer Bankleitzahl, für die Informationen gesucht werden sollen.

Da die gesuchte Bankleitzahl (oder Teil der Bankleitzahl) möglicherweise mehrfach im BLZ-Bestand der Deutschen Bundesbank enthalten ist, muss die Funktion mehrfach aufgerufen werden.

***First*** Geben Sie hier einen Wert > 0 an, wenn Sie die Funktion zum ersten Mal aufrufen. Damit wird die Funktion intern initialisiert. Nach dem ersten Aufruf dieser Funktion wird der Wert von der API automatisch auf 0 gesetzt.

***BLZStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct XMLBLZStruct

```
{   char    BLZ[8+1]          Bankleitzahl
    char    NameKurz[27+1]    Kurzname der Bank
    char    NameLang[35+1]    Langname der Bank
    char    PLZ[5+1]          Postleitzahl
    char    Ort[25+1]         Ort der Bank
    int     EigeneBLZ         Wert=1, wenn BLZ-führende Stelle, Wert=0 wenn Filiale
    char    Verfahren[2+1]    Prüfziffernverfahren lt. Deutscher Bundesbank
    int     DelHinweis        Wert=1, wenn BLZ zur Löschung vorgemerkt ist, sonst 0
```

| | | |
|---|---|---|
| char | ChangeKz | Änderungskennzeichen: A=Neuer Datensatz, D=Löschung, U=Unverändert, M=Modifiziert |
| char | NachfolgeBLZ[8+1] | Nachfolge-BLZ, wenn vorhanden |
| char | BIC[11+1] | Zur BLZ gehörender BIC [1)] |

}

Zusätzliche Erklärungen

1) Bei dem hier zurückgelieferten BIC handelt es sich um den BIC der im Bankleitzahlenbestand der Deutschen Bundesbank hinterlegt ist. Der erforderliche BIC für den SEPA-Zahlungsverkehr kann von diesem BIC abweichen, da er auch noch von anderen Faktoren beeinflusst wird.

   Verwenden Sie für den SEPA-Zahlungsverkehr daher immer den BIC, der mit der Funktion SepaTools_ConvertBLZKonto ermittelt wurde (Bei einigen Banken ist der BIC auch abhängig von einem bestimmten Kontonummernkreis).

## *64.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Daten wurden aus der Datenbank ausgelesen. |
| -1 | Sie sind am Ende der Datenbank angekommen. Es stehen keine weiteren Daten mehr zur Verfügung. |
| -2 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 65 SepaTools_CheckPruefziffer

## *65.1 Zweck der Funktion*

Mit dieser Funktion können Sie eine Kontonummer einer **deutschen Bank** auf ihre Gültigkeit hin überprüfen.

## *65.2 Funktionsaufruf*

**int SepaTools_CheckPruefziffer(const char *BLZ, const char *Konto)**

## *65.3 Parameter*

| | |
|---|---|
| ***BLZ*** | Bitte übergeben Sie hier die 8stellige Bankleitzahl einer deutschen Bank, deren Kontonummer überprüft werden soll. |
| ***Konto*** | Bitte übergeben Sie hier die zu überprüfende Kontonummer. |

## *65.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Die Kontonummer ist gültig. |
| -1 | die Kontonummer ist ungültig. |
| -2 | Die Datenbank BLZ.dat konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (DataPath). |
| -3 | Die übergebene Bankleitzahl wurde im BLZ-Bestand der Deutschen Bundesbank nicht gefunden. |
| -4 | Für österreichische Banken (Bankleitzahl 5stellig) kann keine Prüfziffer berechnet werden. |
| -5 | Es wurde keine Kontonummer oder die Kontonummer 0 übergeben. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 66 SepaTools_GetPurposeCode

## *66.1 Zweck der Funktion*

Über diese Funktion kann ein Purpose Code oder ein Category Purpose Code auf Existenz geprüft werden. Wenn der Code existiert, werden zusätzliche Informationen über den Code zurückgeliefert.

Die verfügbaren Purpose Codes entsprechen den Tabellen aus den ISO External Code Sets.

## *66.2 Funktionsaufruf*

**int SepaTools_GetPurposeCode(const char *Code, PurposeStruct *Purpose, int Flag)**

## *66.3 Parameter*

***Code*** Übergeben Sie hier einen Pointer auf den gesuchten Purpose Code (z.B. BENE, SALA usw.). Der Code muss nullterminiert, 4stellig sein.

***Purpose*** Übergeben Sie hier einen Pointer auf die Struktur ***PurposeStruct***. Die Struktur ist nachfolgend dargestellt.

***Flag*** Übergeben Sie hier ein Flag, das angibt wie eine fehlende deutsche Übersetzung interpretiert werden soll. Dabei sind folgende Werte erlaubt:

0 Bei fehlender Übersetzung wird der Wert **N/A** übergeben.
1 Die fehlende Übersetzung wird mit „Keine Übersetzung“ dargestellt.
2 Die fehlende Übersetzung wird mit dem englischem Originalbegriff dargestellt.

***PurposeStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

```
struct   PurposeStruct

{   int      Prio            Priorität dieses Purpose Codes [1)]
    int      Typ             Der Typ des Purpose Codes [2)]
    char     Code[4+1]       Der 4stellige Purpose Code
    char     Klasse[30+1]    Die Klassifizierung (Englisch) des Purpose Codes
    char     Name[60+1]      Die englische Original Bezeichnung des Codes
    char     Deutsch[60+1]   Die deutsche Übersetzung des Codes (soweit verfügbar)
    char     Reserve[100]    Reservebereich zur späteren Verwendung
}
```

Zusätzliche Erklärungen

1) Durch diesen Code wird die „subjektive“ Wichtigkeit des Purpose Codes dargestellt. Dabei sind folgende Werte möglich.

   0 Dieser Purpose Code wird sehr häufig verwendet, z.B. BENE oder SALA usw.
   1 Die restlichen Purpose Codes erhalten den Wert 1.

2) Purpose Codes werden im Transaktionsteil als Purpose Codes und im Payment Info Teil als Category Purpose Codes verwendet. Dieser Eintrag kann folgende Werte annehmen:

1 Der Code wird **nur** als Purpose Code verwendet.
2 Der Code wird **nur** als Category Purpose Code verwendet.
3 Der Code wird als Purpose Code **und** als Category Purpose Code verwendet.

### *66.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Der Purpose Code wurde ermittelt. Die Ergebnisse in der Struktur können verwendet werden. |
| -1 | Der übergebene Code muss eine Länge von genau 4 Zeichen haben. Dies war nicht der Fall. |
| -2 | Der Purpose Code konnte nicht ermittelt werden. Es wurden keine Daten geliefert. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 67 SepaTools_SearchPurposeCodes

## *67.1 Zweck der Funktion*

Über diese Funktion können die verfügbaren Purpose Codes abgerufen werden. Die verfügbaren Purpose Codes entsprechen den Tabellen aus den ISO External Code Sets.

Um alle Codes auszulesen, muss diese Funktion in einer Schleife solange aufgerufen werden, bis das Ergebnis der Funktion <>0 ist.

## *67.2 Funktionsaufruf*

**int SepaTools_SearchPurposeCodes(*int First, PurposeStruct *Purpose, int Flag, int Typ)**

***First*** Geben Sie hier einen Wert (pointer) > 0 an, wenn Sie die Funktion zum ersten Mal aufrufen. Damit wird die Funktion intern initialisiert. Nach dem ersten Aufruf dieser Funktion wird der Wert von der API automatisch auf 0 gesetzt.

***PurposeStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

struct PurposeStruct

| | | | |
|---|---|---|---|
| { | int | Prio | Priorität dieses Purpose Codes [1)] |
| | int | Typ | Der Typ des Purpose Codes [2)] |
| | char | Code[4+1] | Der 4stellige Purpose Code |
| | char | Klasse[30+1] | Die Klassifizierung (Englisch) des Purpose Codes |
| | char | Name[60+1] | Die englische Original Bezeichnung des Codes |
| | char | Deutsch[60+1] | Die deutsche Übersetzung des Codes (soweit verfügbar) |
| | char | Reserve[100] | Reservebereich zur späteren Verwendung |
| } | | | |

***Flag*** Übergeben Sie hier ein Flag, das angibt wie eine fehlende deutsche Übersetzung interpretiert werden soll. Dabei sind folgende Werte erlaubt:

0 Bei fehlender Übersetzung wird der Wert **N/A** übergeben.
1 Die fehlende Übersetzung wird mit „Keine Übersetzung“ dargestellt.
2 Die fehlende Übersetzung wird mit dem englischem Originalbegriff dargestellt.

***Typ*** Über geben Sie hier einen Wert der angibt, welche Arten der Codes ausgelesen werden sollen. Dabei sind folgende Werte erlaubt:

1 Es werden **nur** Purpose Codes ausgelesen.
2 Es werden **nur** Category Purpose Codes ausgelesen.
3 Es werden Purpose Codes und Category Prupose Codes ausgelesen.

Zusätzliche Erklärungen

1) Durch diesen Code wird die „subjektive“ Wichtigkeit des Purpose Codes dargestellt. Dabei sind folgende Werte möglich.

0 Dieser Purpose Code wird sehr häufig verwendet, z.B. BENE oder SALA usw.
1 Die restlichen Purpose Codes erhalten den Wert 1.

2) Purpose Codes werden im Transaktionsteil als Purpose Codes und im Payment Info Teil als Category Purpose Codes verwendet. Dieser Eintrag kann folgende Werte annehmen:

1 Der Code wird **nur** als Purpose Code verwendet.
2 Der Code wird **nur** als Category Purpose Code verwendet.
3 Der Code wird als Purpose Code **und** als Category Purpose Code verwendet.

### *67.3 Returncodes*

0 Alles verlief erfolgreich. Der Purpose Code wurde ermittelt. Die Ergebnisse in der Struktur können verwendet werden.

-1 Es stehen keine weiteren Codes mehr zur Verfügung.

-2 Die Datenbank Sepa.dat konnte nicht geöffnet werden.

-3 Das Flag für die fehlende Übersetzung darf nur die Werte 0, 1 oder 2 haben.

-4 Der Typ des Codes darf nur die Werte 1, 2 oder 3 haben.

# 68 SepaTools_GetReasonCode

## *68.1 Zweck der Funktion*

Über diese Funktion kann ein Reason Code auf Existenz geprüft werden. Wenn der Code existiert, werden zusätzliche Informationen über den Code zurückgeliefert.

Die verfügbaren Reason Codes entsprechen den Tabellen aus den ISO External Code Sets.

## *68.2 Funktionsaufruf*

**int SepaTools_GetReasonCode(const char *Code, ReasonStruct *Reason)**

## *68.3 Parameter*

***Code*** Übergeben Sie hier einen Pointer auf den gesuchten Purpose Code (AB05, AM04 usw.). Der Code muss nullterminiert, 4stellig sein.

***Reason*** Übergeben Sie hier einen Pointer auf die Struktur Reason***Struct***. Die Struktur ist nachfolgend dargestellt.

***ReasonStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

```
struct   ReasonStruct

{   char    Code[4+1]       Der 4stellige Reason Code
    char    TSErg[3+1]      Optionale Textschlüssel ergänzung 1)
    char    Name[100+1]     Die Bezeichnung des Codes
    char    Reserve[100]    Reservebereich zur späteren Verwendung
}
```

Zusätzliche Erklärungen

1) Für einige (allerdings die wenigsten) Reasen Codes gibt es äquivalente Entsprechnungen zu den früheren (alten) Textschlüssel Ergänzungen aus dem DTA Bereich.

   Sollte es eine solche Entsprechnung geben, so wird dieses Feld gefüllt.

## *68.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Der Reason Code wurde ermittelt. Die Ergebnisse in der Struktur können verwendet werden. |
| -1 | Der übergebene Code muss eine Länge von genau 4 Zeichen haben. Dies war nicht der Fall. |
| -2 | Der Reason Code konnte nicht ermittelt werden. Es wurden keine Daten geliefert. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 69 SepaTools_SearchReasonCodes

## *69.1 Zweck der Funktion*

Über diese Funktion können die verfügbaren Reason Codes abgerufen werden. Die verfügbaren Reason Codes entsprechen den Tabellen aus den ISO External Code Sets.

Um alle Codes auszulesen, muss diese Funktion in einer Schleife solange aufgerufen werden, bis das Ergebnis der Funktion <>0 ist.

## *69.2 Funktionsaufruf*

**int SepaTools_SearchReasonCodes(*int First, ReasonStruct *Reason, int Typ)**

***First*** Geben Sie hier einen Wert (pointer) > 0 an, wenn Sie die Funktion zum ersten Mal aufrufen. Damit wird die Funktion intern initialisiert. Nach dem ersten Aufruf dieser Funktion wird der Wert von der API automatisch auf 0 gesetzt.

***ReasonStruct*** In dieser Struktur werden die Ergebnisse zurückgeliefert.

```
struct   ReasonStruct

{   char    Code[4+1]       Der 4stellige Reason Code
    char    TSErg[3+1]      Optionale Textschlüssel Ergänzung 1)
    char    Name[100+1]     Die Bezeichnung des Codes
    char    Reserve[100]    Reservebereich zur späteren Verwendung
}
```

***Typ*** In den External Code Sets sind unterschiedlche Reason Codes definiert. Insgesamt sind es mehr als 350. Einzelne Reason Codes können auch in unterschiedlichen Geschäftsvorfällen vorkommen.

Die Datei mit den External Code Sets finden Sie auf https://sepa-tools.de und dort unter dem Menüpunkt „Weitere Infos“.

Der Wert **Typ** ist Bit codiert. Damit wird ausgewählt, welche Reason Codes ausggegeben werden.

Die nachfolgende Tabelle zeigt beispielhaft die Kontanten die hierzu verwendet werden können.

| *Konstante* | *Wert* | *Originalbezeichnung innerhalb der External Code Sets mit Angabe der Excel Blatt Nummer* |
|---|---|---|
| Reason_None | 0 | Es werden alle Reason Codes ausgegeben. |
| Reason_Mandat | 1 | 8-MandateReason |
| Reason_Return | 2 | 13-ReturnReason |
| Reason_Reversal | 4 | 14-ReversalReason |
| Reason_Status | 8 | 16-StatusReason |
| Reason_Verification | 16 | 18-VerificationReason |

| *Konstante* | *Wert* | *Originalbezeichnung innerhalb der External Code Sets mit Angabe der Excel Blatt Nummer* |
|---|---|---|
| Reason_RePresentation | 32 | 34-RePresentmentReason |
| Reason_Contract | 64 | 54-ContractClosureReason |
| Reason_Received | 128 | 60-ReceivedReason |
| Reason_Accepted | 256 | 61-AcceptedReason |
| Reason_Pending | 512 | 62-PendingProcessingReason |
| Reason_Rejected | 1024 | 63-RejectedReason |
| Reason_Cancel | 2048 | 66-CancellationReason |
| Reason_MandatSuspended | 4096 | 68-MandateSuspensionReason |
| Reason_PaymentCompensation | 8192 | 79-PaymentCompensationReason |
| Reason_CredAmendment | 16384 | 93-CREnrolmentAmendmentReason |
| Reason_DbtrAmendment | 32768 | 94-DbtrActAmendmentReason |
| Reason_CredCancel | 65536 | 95-CREnrolmentCancelReason |
| Reason_DbtrCancel | 131072 | 96-DbtrActCancellationReason |
| Reason_CredStatus | 262144 | 97-CREnrolmentStatusReason |
| Reason_Dbtr_Status | 524288 | 98-DbtrActivationStatusReason |

Zusätzliche Erklärungen

1) Für einige (allerdings die wenigsten) Reasen Codes gibt es äquivalente Entsprechnungen zu den früheren (alten) Textschlüssel Ergänzungen aus dem DTA Bereich.

### *69.3 Returncodes*

0 Alles verlief erfolgreich. Der Reason Code wurde ermittelt. Die Ergebnisse in der Struktur können verwendet werden.

-1 Es stehen keine weiteren Codes mehr zur Verfügung.

-2 Die Datenbank Sepa.dat konnte nicht geöffnet werden.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 70 SepaTools_AddUserBIC

## *70.1 Zweck der Funktion*

Über diese Funktion können in den User-Daten Bankleitzahlen hinterlegt werden, für die ein vom BLZ-Verzeichnis der Deutschen Bundesbank oder der Österreichischen Nationalbank abweichender BIC hinterlegt werden soll. Bei der Umsetzung von Bankleitzahlen in BICs wird dann dieser abweichende BIC verwendet.

Die so gespeicherten Daten werden nur bei der Umsetzung von BLZ und Kontonummer in BIC und IBAN verwendet. Dies ist konkret der Fall bei den Funktionen:

SepaTools_CreateXML (und den dazugehörenden Funktionen)
SepaTools_ConvertBLZKonto
SepaTools_ReadDTA (und den dazugehörenden Funktionen).

## *70.2 Funktionsaufruf*

**int SepaTools_AddUserBIC(const char *BLZ, const char *BIC)**

## *70.3 Parameter*

| | |
|---|---|
| ***BLZ*** | Übergeben Sie bitte hier einen Pointer auf die Zeichenkette, die die Bankleitzahl (8stellig für Deutschland oder 5stellig für Österreich) enthält. |
| ***BIC*** | Übergeben Sie bitte hier einen Zeiger auf den BIC, der verwendet werden soll, wenn die angegebene Bankleitzahl umgesetzt werden soll. |

## *70.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die übergebene Bankleitzahl ist formal falsch, Länge <>5 und <>8. |
| -2 | Der übergebene BIC ist formal falsche, Länge <>8 oder Länge <>11. |
| -5 | Der Eintrag konnte nicht in die Datenbank geschrieben werden, da der Eintrag bereits in der Datenbank vorhanden ist. |
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 71 SepaTools_DelUserBIC

## *71.1 Zweck der Funktion*

Über diese Funktion wird eine Kombination aus Bankleitzahl und BIC aus der Datenbank für die User-Daten gelöscht.

## *71.2 Funktionsaufruf*

**int SepaTools_DelUserBIC(const char *BLZ)**

## *71.3 Parameter*

| | |
|---|---|
| ***BLZ*** | Übergeben Sie bitte hier einen Pointer auf die Zeichenkette, die die Bankleitzahl (8stellig für Deutschland und 5stellig für Österreich) enthält. |

## *71.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die übergebene Bankleitzahl ist formal falsch, Länge <>5 und <>8. |
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 72 SepaTools_GetUserBIC

## *72.1 Zweck der Funktion*

Über diese Funktion kann der zu der korrespondierenden Bankleitzahl gespeicherte BIC wieder ausgelesen werden.

## *72.2 Funktionsaufruf*

**int SepaTools_GetUserBIC(const char *BLZ, char *BIC)**

## *72.3 Parameter*

| | |
|---|---|
| ***BLZ*** | Übergeben Sie bitte hier einen Pointer auf die Zeichenkette, die die Bankleitzahl (8stellig für Deutschland und 5stellig für Österreich) enthält. |
| ***BIC*** | In dieser Variablen wird der korrespondierende BIC zurückgegeben. Bitte beachten Sie, dass der Speicherbereich ausreichend groß ist und mindestens 11+1 Zeichen (incl. der abschließenden 0) aufnehmen kann. |

## *72.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die übergebene Bankleitzahl ist formal falsch, Länge <>8 und <>5. |
| -6 | Zu der angegebenen Bankleitzahl wurde kein Eintrag gefunden. |
| -7 | Der Eintrag konnte aus technischen Gründen aus der Datenbank nicht ausgelesen werden. |
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 73 SepaTools_FindUserBIC

## *73.1 Zweck der Funktion*

Diese Funktion dient zum generellen Auslesen der Datenbank. Ein wiederholter Aufruf dieser Funktion bis der Returncode ungleich 0 ist, führt zum Auslesen aller Kombinationen von Bankleitzahl und BIC.

## *73.2 Funktionsaufruf*

**int SepaTools_FindUserBIC(char *BLZ, char *BIC, int *First)**

## *73.3 Parameter*

| | |
|---|---|
| ***BLZ*** | In dieser Variablen wird die ausgelesene Bankleitzahl (8stellig für Deutschland und 5stellig für Österreich) übergeben. Die Variable muss genügend groß sein und mindestens 8+1 Zeichen (incl. der abschließenden 0) aufnehmen können. |
| ***BIC*** | In dieser Variablen wird der korrespondierende BIC zurückgegeben. Bitte beachten Sie, dass der Speicherbereich ausreichend groß ist und mindestens 11+1 Zeichen (incl. der abschließenden 0) aufnehmen kann. |
| ***First*** | Diese Variable ist beim ersten Aufruf dieser Funktion mit einem Wert <> 0 zu belegen. Damit wird die interne Aufrufsequenz initialisiert.<br><br>Die API setzt im Rahmen des ersten Funktionsaufrufs diese Variable automatisch auf 0. |

## *73.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -6 | Sie sind am Ende der Datenbank angekommen. Es liegen keine weiteren Einträge mehr vor. |
| -7 | Der Eintrag konnte aus technischen Gründen aus der Datenbank nicht ausgelesen werden. |
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 74 SepaTools_AddUserIBAN

## *74.1 Zweck der Funktion*

Über diese Funktion können in den User-Daten Bankleitzahlen und dazugehörende Kontonummern hinterlegt werden, für die eine abweichende IBAN ermittelt werden soll. Ebenso kann ein abweichender BIC hinterlegt werden.

Ein abweichender BIC hat dabei nur auf diese Kombination von Bankleitzahl und Kontonummer Einfluss. Soll global für eine Bankleitzahl ein abweichender BIC verwendet werden, so benutzen Sie bitte die Funktion SepaTools_AddUserBIC.

Die so gespeicherten Daten werden nur bei der Umsetzung von BLZ und Kontonummer in BIC und IBAN verwendet. Dies ist konkret der Fall bei den Funktionen:

SepaTools_CreateXML (und den dazugehörenden Funktionen)
SepaTools_ConvertBLZKonto
SepaTools_ReadDTA (und den dazugehörenden Funktionen).

## *74.2 Funktionsaufruf*

**int SepaTools_AddUserIBAN(const XMLConvertStruct *ConvertStruct)**

## *74.3 Parameter*

***ConvertStruct*** In dieser Struktur werden die Werte übergeben. Es müssen mit Ausnahme des BICs alle Felder gefüllt werden. Das Feld BIC ist nur zu füllen, wenn auch für die Kombination aus der übergebenen Bankleitzahl und der Kontonummer auch ein abweichender BIC verwendet werden soll.

```
struct   XMLConvertStruct

{   char    BLZ[8+1]        Bankleitzahl (8stellig für Deutschland und 5stellig für Österreich) , die verwendet werden soll.
    char    Konto[11+1]     Kontonummer, die verwendet werden soll.
    char    BIC[11+1]       Optional der zu ersetzende BIC.
    char    IBAN[35+1]      Die zu ersetzende IBAN.
}
```

## *74.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die übergebene Bankleitzahl ist formal falsch, Länge <>5 und <>8. |
| -2 | Der übergebene BIC ist formal falsche, Länge <>8 oder Länge <>11. |
| -3 | Die übergebene Kontonummer ist formal falsch. Der nummerische Wert der Kontonummer ist kleiner als 10. |

| | |
|---|---|
| -4 | Es wurde entweder keine IBAN, oder eine IBAN mit ungültiger Prüfzahl übergeben. |
| -5 | Der Eintrag konnte nicht in die Datenbank geschrieben werden, da der Eintrag bereits in der Datenbank vorhanden ist. |
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 75 SepaTools_DelUserIBAN

## *75.1 Zweck der Funktion*

Über diese Funktion wird der über die Kombination aus Bankleitzahl und Kontonummer dargestellte Eintrag in der Datenbank gelöscht.

## *75.2 Funktionsaufruf*

**int SepaTools_DelUserIBAN(const XMLConvertStruct *ConvertStruct)**

## *75.3 Parameter*

***ConvertStruct*** In dieser Struktur werden die Werte übergeben. Es reicht aus, wenn die Bankleitzahl und die Kontonummer übergeben werden. BIC und IBAN werden für diese Funktion nicht benötigt und auch nicht überprüft.

```
struct   XMLConvertStruct

{   char    BLZ[8+1]          Bankleitzahl (8stellig für Deutschland und 5stellig für Österreich), die verwendet werden soll.
    char    Konto[11+1]       Kontonummer, die verwendet werden soll.
    char    BIC[11+1]         Keine Angabe erforderlich.
    char    IBAN[35+1]        Keine Angabe erforderlich.
}
```

## *75.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die übergebene Bankleitzahl ist formal falsch, Länge <>5 und <>8. |
| -3 | Die übergebene Kontonummer ist formal falsch. Der nummerische Wert der Kontonummer ist kleiner als 10. |

| | |
|---|---|
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 76 SepaTools_GetUserIBAN

## *76.1 Zweck der Funktion*

Über diese Funktion wird ein Datensatz mit Bankleitzahl, Kontonummer zu ersetzende IBAN und u.U. zu ersetzenden BIC aus der Datenbank der User-Daten ausgelesen.

## *76.2 Funktionsaufruf*

**int SepaTools_GetUserIBAN(XMLConvertStruct *ConvertStruct)**

## *76.3 Parameter*

***ConvertStruct*** In dieser Struktur werden die Werte übergeben und die ausgelesenen Werte zurückgegeben. Es reicht aus, wenn die Bankleitzahl und die Kontonummer übergeben werden. BIC und IBAN werden dann zurückgeliefert.

struct XMLConvertStruct

| | | | |
|---|---|---|---|
| { | char | BLZ[8+1] | Bankleitzahl (8stellig für Deutschland und 5stellig für Österreich), die verwendet werden soll. |
| | char | Konto[11+1] | Kontonummer, die verwendet werden soll. |
| | char | BIC[11+1] | Der gespeicherte BIC. |
| | char | IBAN[35+1] | Die gespeicherte IBAN. |
| } | | | |

## *76.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die übergebene Bankleitzahl ist formal falsch, Länge <>5 und <>8. |
| -3 | Die übergebene Kontonummer ist formal falsch. Der nummerische Wert der Kontonummer ist kleiner als 10. |
| -6 | Zu der angegebenen Bankleitzahl und der angegebenen Kontonummer wurde kein Eintrag in der Datenbank gefunden. |
| -7 | Das Lesen des Datensatzes ist aus technischen Gründen fehlgeschlagen. |
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 77 SepaTools_FindUserIBAN

## *77.1 Zweck der Funktion*

Diese Funktion dient zum generellen Auslesen der Datenbank. Ein wiederholter Aufruf dieser Funktion bis der Returncode ungleich 0 ist, führt zum Auslesen aller Kombinationen von Bankleitzahl, Kontonummer, IBAN und BIC.

## *77.2 Funktionsaufruf*

**int SepaTools_FindUserIBAN(const XMLConvertStruct *ConvertStruct , int *First)**

## *77.3 Parameter*

***ConvertStruct*** In dieser Struktur werden die ausgelesenen Werte zurückgegeben. Es brauchen keine Werte übergeben werden. Sinnvollerweise sollte die Struktur mit NULL initialisiert werden.

```
struct    XMLConvertStruct

{   char    BLZ[8+1]          Die gespeicherte Bankleitzahl (8stellig für Deutschland und
                              5stellig für Österreich).
    char    Konto[11+1]       Die gespeicherte Kontonummer.
    char    BIC[11+1]         Der gespeicherte BIC.
    char    IBAN[35+1]        Die gespeicherte IBAN.
}
```

***First*** Diese Variable ist beim ersten Aufruf dieser Funktion mit einem Wert <> 0 zu belegen. Damit wird die interne Aufrufsequenz initialisiert.

Die API setzt im Rahmen des ersten Funktionsaufrufs diese Variable automatisch auf 0.

## *77.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -6 | Sie sind am Ende der Datenbank angekommen. Es stehen in der Datenbank keine weiteren Einträge mehr zur Verfügung. |
| -7 | Das Lesen des Datensatzes ist aus technischen Gründen fehlgeschlagen. |
| -9 | Die Verarbeitung der User-Daten wurde nicht initialisiert. Bitte überprüfen Sie den übergebenen Wert für UserPath bei der Funktion SepaTools_Init. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 78 SepaTools_GetTargetDatum

## *78.1 Zweck der Funktion*

Buchungen im SEPA-Raum können nur an einem der so genannten Target-Tage erfolgen. Dies sind die Tage, an denen das europaweite Clearingsystem arbeitet. Target-Tage sind alle Tage des Jahres mit Ausnahme von Samstagen und Sonntagen, 25. und 26. Dezember, 1. Januar, 1. Mai, Karfreitag und Ostermontag.

Die Funktion kann auch den Karfreitag und den Ostermontag nach dem Kalender der westlichen Kirchen berechnen (orthodoxe Kirchen andere Berechnung, auf Anfrage möglich).

Wenn Sie der Funktion ein Basisdatum (z.B. Tagesdatum) übergeben, so wird unter Berücksichtigung von Zusatztagen das nächste mögliche Datum ermittelt, das ein Target-Tag ist (Parameter Skip=0). Dies ist hilfreich, wenn es sich um das Fälligkeitsdatum von Lastschriften unter Einhaltung der Vorlaufzeiten handelt. Fällt das errechnete Datum auf einen „Nicht Target-Tag“, so wird das Datum bis zum nächsten Target-Tag hochgerechnet.

Wird dagegen der Wert von Skip auf einen Wert <> 0 gesetzt, so werden bei der Berechnung der im Parameter Tage übergebenen Zusatztage die „Nicht Target-Tage“ übersprungen. Bei der Berechnung der Vorlauffristen werden „Nicht Target-Tage nicht mitgezählt.

Das Ergebnis wird in einer Zeichenkette zurückgegeben.

Beispiel:

Tagesdatum ist der 19. Dezember 2012. Das Buchungsdatum soll 6 Tage in der Zukunft liegen. Der Parameter Skip wurde mit 0 belegt. Das nächste Target-Datum ist damit der 27.122012.

Wird der Parameter Skip allerdings mit einem Wert <> 0 belegt, so werden bei der Zählung der 6 Tage die „Nicht Target-Tage“ übersprungen. Damit ergibt sich als Zieldatum der 31.12.2012.

Um diese Funktion zu testen, sollten Sie sich einen Kalender daneben legen.

## *78.2 Funktionsaufruf*

```
int SepaTools_GetTargetDatum(const char *Datum,
                             const int Tage,
                             const Int Skip,
                             char *TargetDatum)
```

## *78.3 Parameter*

***Datum*** Bitte übergeben Sie hier einen Zeiger auf eine Zeichenkette, die das Ursprungsdatum im Format TTMMJJJJ (im Beispiel der 19.12.2012) darstellt.

***Tage*** Übergeben Sie hier bitte die Anzahl Tage, um die das Ursprungsdatum verlängert werden soll. Gültige eingaben liegen zwischen 0 und 1000. Wird hier der Wert 0 verwendet, so liefert die Funktion unabhängig vom Wert Skip das gleiche Ergebnis.

***Skip***

**0** Wird hier der Wert 0 übergeben, so werden bei der Zählung der Zusatztage die „Nicht Target-Tage“ nicht übersprungen.

**1** Bei einem Wert von 1 werden die „Nicht Target-Tage“ bei der Zählung der zusätzlichen Tage übersprungen.

**2** Bei einem Wert von 2 werden die „Nicht Target-Tage“ bei der Zählung der zusätzlichen Tage ebenfalls übersprungen.

Wenn das übergebene Datum auf einen Nicht-Target-Tag fällt (z.B. auf einen Sonntag), so wird das Datum auf den nächsten Target-Tag hochgerechnet.

***TargetDatum***

Dies ist das ermittelte Datum (im Format TTMMJJJJ), das garantiert einen Target-Tag darstellt. Bitte übergeben Sie hier einen Zeiger auf einen Speicherbereich, der mindesten 9 Zeichen aufnehmen kann (8 Zeichen für das Datum im Format TTMMJJJJ und ein Zeichen für die abschließende NULL).

### *78.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Das Target-Datum wurde ermittelt. |
| -1 | Das übergebene Ursprungsdatum ist formal ungültig. |
| -2 | Das übergebene Ursprungsdatum ist kleiner als der 1. Januar 2000. Erst ab diesem Datum funktioniert die Funktion |
| -3 | Der übergebene Wert für Tage ist kleiner als 0. |
| -4 | Der übergebene Wert für Tage ist größer als 1000. |
| -5 | Das Target-Datum konnte nicht ermittelt werden. Dies würde auf einen Berechnungsfehler in der Logik hindeuten. Dieser Fehlercode dürfte nie auftreten. Wenn ja, so informieren Sie bitte den Hersteller. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 79 SepaTools_IsTargetDatum

## *79.1 Zweck der Funktion*

Buchungen im SEPA-Raum können nur an einem der so genannten Target-Tage erfolgen. Dies sind die Tage, an denen das europaweite Clearingsystem arbeitet. Target-Tage sind alle Tage des Jahres mit Ausnahme von Samstagen und Sonntagen, 25. und 26. Dezember, 1. Januar, 1. Mai, Karfreitag und Ostermontag.

Die Funktion kann auch den Karfreitag und den Ostermontag nach dem Kalender der westlichen Kirchen berechnen.

Diese Funktion prüft, ob das übergebene Datum ein gültiges Target-Datum ist.

## *79.2 Funktionsaufruf*

**int SepaTools_IsTargetDatum(const char *Datum)**

## *79.3 Parameter*

| | |
|---|---|
| ***Datum*** | Bitte übergeben Sie hier einen Zeiger auf eine Zeichenkette, die das Ursprungsdatum im Format TTMMJJJJ darstellt. Es wird geprüft, ob es sich um ein gültiges Target-Datum handelt. |

## *79.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. Bei dem übergebenen Datum handelt es sich um ein gültiges Target-Datum. |
| -1 | Das übergebene Ursprungsdatum ist formal ungültig. |
| -2 | Das übergebene Ursprungsdatum ist kleiner als der 1. Januar 2000. Erst ab diesem Datum funktioniert die Funktion |
| -3 | Das übergebene Datum ist kein gültiges Target-Datum |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 80 Pain.007-Dateien erzeugen

Mit der API können pain.007-Dateien erzeugt werden. Diese Dateiart dient zum Rückruf von Lastschriften (z.B. wegen Doppeleinreichung). Diese Funktionalität wird grundsätzlich ab November 2017 unterstützt.

Die Unterstützung von pain.007-Dateien durch die Zahlungsdienstleister ist optional. Im Gegensatz zur Verarbeitung von XML-Zahlungsverkehrsdateien gibt es für diese Dateiart keine Verpflichtung der Banken.

Dazu sind drei Funktionen erforderlich, die nachfolgend dargestellt sind. Der logische Ablauf stellt sich folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateXML007 | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_WriteXML007 | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages (Rückrufauftrages) übergeben. |
| … | |
| - Aufruf von SepaTools_CloseXML007 | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |

Innerhalb dieser Funktionen finden umfangreiche Plausibilitätsprüfungen statt.

# 81 SepaTools_CreateXML007

## *81.1 Zweck der Funktion*

Mit dieser Funktion wird die Ausgabe von XML-Dateien initialisiert. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

## *81.2 Funktionsaufruf*

**int SepaTools_CreateXML007( XMLCreate007Struct *CreateStruct)**

## *81.3 Parameter*

***CreateStruct*** Hier wird ein Pointer auf die Struktur CreateStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für SEPA-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

struct XMLCreate007Struct

| | | | |
|---|---|---|---|
| { | char | ExportPfad[255+1] | Laufwerk und Pfad für die Ausgabe der XML-Datei. |
| | char | XMLName[35+1] | Der Name der zu erzeugenden XML-Datei [1]. |
| | char | MsgId[35+1] | Message-Id (Kennung) für die XML-Datei [2]. |
| | char | OrgMsgId[35+1] | Die ursprüngliche Msg-Id der zurückzurufenden Lastschriften |
| | char | EinreicherName[70+1] | Name des Einreichers der Datei (mind. 3 Zeichen). |
| | char | EinreicherBIC[11+1] | Optionaler BIC des Einreichers der Rückrufe. |
| | char | Reserve[1000] | Reserve zur späteren Verwendung. |
| } | | | |

Zusätzliche Erklärungen:

1) Unabhängig von der übergebenen oder nicht übergebenen Dateinamenserweiterung wird die Dateinamenserweiterung immer auf .xml gesetzt.

   Wird hier ein Leerstring oder der Wert „Dummy.xml“ übergeben, so verwaltet die API den Dateienamen intern. Dies ist erforderlich, wenn die XML-Datei als Byte-Array im Speicher ausgegeben werden soll.

2) Die Message-Id soll eine möglichst eindeutige Identifizierung der XML-Datei sein. Wenn Sie hier einen Leerstring übergeben, so wird intern automatisch eine Message-Id gebildet und in der Struktur zurückgegeben.

### *81.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Es wurde ein formal unrichtiger Pfad (Länge < 2 Zeichen) für die Exportdatei übergeben. |
| -2 | Das angegebene Verzeichnis für die Exportdatei existiert nicht. |
| -3 | Der Datenträger, der im Pfadnamen der Exportdatei angegeben wurde ist schreibgeschützt. Die Datei kann nicht geschrieben werden. |
| -4 | Der Name des Einreichers der Daten wurde zu kurz (< 3 Zeichen) angegeben. |
| -5 | Die zum Ablauf der Funktion erforderliche temporäre Datei konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (TempPath). |
| -6 | Der Pfadname für die Exportdatei ist zu lange (>200 Zeichen). |
| -7 | Das angegebene Laufwerk im Pfad für die Exportdatei ist nicht bereit (möglicherweise kein Datenträger eingelegt). |
| -8 | Der Name für die XML-Datei wurde zu kurz (< 3 Zeichen) angegeben. |
| -9 | Die Funktion SepaTools_CreateXML wurde bereits aufgerufen. Vor einem weiteren Aufruf muss zuvor die Funktion SepaTools_CloseXML aufgerufen werden.<br><br>Die Funktionalitäten zur Erzeugung einer XML-Datei sind nicht „Multi-Tread“ fähig! |
| -10 | Es wurde keine „ursprüngliche“ Message-Id übergeben. Die Message-Id der ursprünglichen Lastschrftsdatei ist zwingend erforderlich. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 82 SepaTools_WriteXML007

## *82.1 Zweck der Funktion*

Über jeden Aufruf dieser Funktion wird genau ein Datensatz für einen Rückruf übergeben. Es werden umfangreiche Plausibilitätsprüfungen durchgeführt.

Im Wesentlichen werden hier wesentliche Inforamtionen aus den Lastschriftsdaten der Ursprungsdatei übergeben.

## *82.2 Funktionsaufruf*

**int SepaTools_WriteXML007( XMLWrite007Struct *WriteStruct)**

## *82.3 Parameter*

***WriteStruct*** Hier wird ein Pointer auf die Struktur WriteStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz für SEPA-Zahlungen handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

struct XMLWrite007Struct

| | | | |
|---|---|---|---|
| { | char | PmInfoId[35+1] | Payment-Info-Id der ursprünglichen Lastschrift. [1)] |
| | int | PmAnzahl | Die Anzahl Datensätze in diesem Payment-Information Block |
| | char | PmSumme[12+1] | Die Gesamtsumme im Payment-Information Block [2)] |
| | int | Batch | Die zurückger. Datensätze einzeln buchen werden. [3)] |
| | char | Betrag[12+1] | Betrag der ursprünglichen Lastschrift in Cent |
| | char | Reason[4+1] | Grund des Rückrufs [4)] |
| | char | AufDatum[8+1] | Auftragsdatum der ursprünglichen Lastschrift [5)] |
| | char | CI[35+1] | CI der ursprünglichen Lastschrift |
| | int | B2B | 0=CORE Lastschrift, 1=B2B Lastschrift (ursprünglich) |
| | char | SequenceType[4+1] | Sequenz der ursprünglichen Lastschrift [6)] |
| | char | MandatId[35]+1 | Mandats-Id der ursprünglichen Lastschrift |
| | char | MandatDatum[8+1] | Mandatsdatum der urspr. Lastschrift (TTMMJJJJ) |
| | char | Reserve[2000] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Hier wird die Payment-Info-Id der ursprünglichen Lastschrift übergeben. In der ursprünglichen Lastschriftsdatei können mehrere Payment-Info Blocks enthalten sein. Hier ist die Payment-Info-Id des Blocks gemeint, der sich auf die zurückzurufende Lastschrift bezieht.

2) Die Gesamtsumme innerhalb des ursprünglichen Payment-Informations Blocks wird in Cent übergeben. Der Betrag 1.876,45 € wird als Zeichenkette 187645 übergeben.

3) Nur wenn eine entsprechende Vereinbarung für Einzelbuchungen mit dem Kunden vorliegt, wird im Falle Batch=1 jeder Reversal einzeln auf dem Kontoauszug des ursprünglichen Lastschrifteinreichers dargestellt.

   Wird Batch=2 übergeben, so erfolgt eine Sammelbuchung.

   Wird der Wert 0 oder kein Wert übergeben, so gelten die Voreinstellungen der Bank.

4) Angabe des Rückrufgrundes. Derzeit sind nur folgende Werte erlaubt:

   **AM05** Doppeleinreichung
   **MS02** Grund nicht bekannt

5) Auftragsdatum, sprich Fälligkeitsdatum der ursprünglichen Lastschrift im Format TTMMJJJJ.

6) Erlaubt sind die Werte FRST, RCUR, OOF und FNAL

### *82.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Das Schreiben des Datensatzes in die temporäre Datei ist gescheitert. |
| -2 | Die Funktion SepaTools_CreateXML007 wurde noch nicht aufgerufen. |
| -101 | Es wurde keine ursprüngliche Payment-Information übergeben. |
| -102 | Die Anzahl der ursprünglichen Transaktionen wurde nicht übergeben. |
| -103 | Die Summe der Beträge der ursprünglichen Transaktionen wurde nicht übergeben. |
| -104 | Es wurde keine ursprüngliche EndeToEnd-Id übergeben. |
| -105 | Es wurde kein ursprünglicher Betrag übergeben. |
| -106 | Es wurde kein Rückrufgrund angegeben. |
| -107 | Es wurde ursprüngliches Ausführungsdatum angegeben. |
| -108 | Das ursprüngliche Ausführungsdatum kann nicht in der Zukunft liegen. |
| -109 | Es wurde keine ursprüngliche CI übergeben. |
| -110 | Der Wert für B2B kann nur 0 oder 1 lauten. |
| -111 | Es wurde kein Wert für die ursprüngliche Sequenz übergeben. |
| -112 | Es wurde ein ungültiger Wert für die ursprüngliche Sequenz übergeben. |
| -113 | Es wurde kein ursprüngliches Mandat übergeben. |
| -114 | Es wurde kein Datum für das ursprüngliche Mandat übergeben oder das Datum ist ungültig. |

| | |
|---|---|
| -115 | Das ursprüngliche Datum des Mandats kann nicht in der Zukunft liegen. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 83 SepaTools_CloseXML007

## *83.1 Zweck der Funktion*

Mit dieser Funktion wird die eigentliche XML-Datei aus der von SepaTools_WriteXML007 erzeugten temporären Datei erzeugt und in die definierte Datei geschrieben.

## *83.2 Funktionsaufruf*

**int SepaTools_CloseXML007( XMLCloseStruct *CloseStruct)**

## *83.3 Parameter*

***CloseStruct*** Hier wird ein Pointer auf die Struktur CloseStruct übergeben. Die Struktur ist nachfolgend dargestellt. Die Struktur wird leer übergeben und von der API mit Informationen für die Applikation gefüllt.

```
struct   XMLCloseStruct

{  int     Anzahl                Anzahl der erzeugten Datensätze.
   char    SummeBetrag[12+1]     Gesamtsumme der Beträge in der XML-Datei als
                                 Zeichenkette in Cent.
}
```

## *83.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die XML-Datei konnte nicht erzeugt/initialisiert werden. Bitte setzen Sie sich mit dem Hersteller in Verbindung. |
| -2 | Beim Erzeugen der einzelnen Datensätze ist ein Fehler aufgetreten. Bitte setzen Sie sich mit dem Hersteller in Verbindung. |
| -3 | Beim Schreiben der XML-Datei auf den Datenträger ist ein Fehler aufgetreten. |

Die folgenden Returncodes -4 bis -7 können nur auftreten, wenn im „Speichermodus“ gearbeitet wird (es wurde kein Dateiname für die XML-Datei übergeben). Die XML-Datei wird nicht als Datei, sondern über die Funktion SepaTools_XMLGetData im Speicher übergeben.

| | |
|---|---|
| -4 | Es stehen keine Daten zur Ausgabe der XML-Datei im Speicher zur Verfügung. |
| -5 | Die Größe der XML-Daten ist größer als die maximal erlaubte Größe von derzeit 500 MB. |
| -6 | Der Speicherbereich zu Übergabe der Daten konnte nicht belegt werden. Möglicherweise steht zu wenig Hauptspeicher zur Verfügung. |
| -7 | Die Ausgabe der XML-Daten in den Speicherbreich ist fehlgeschlagen. |

| | |
|---|---|
| -8 | Es konnten keine Datensätze geschrieben werden, da alle Datensätze fehlerhaft waren. |
| -9 | Die temporäre Datei konnte nicht geöffnet werden. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

## 84 CGI-Dateien erzeugen

Mit der API können CGI-Dateien erzeugt werden. Das CGI-Format ist ein, gegenüber SEPA deutlich erweitertes XML-Format. Über diesen Weg können im XML-Format Auslandszahlungsaufträge auch außerhalb des SEPA-Raums erteilt werden.

Allerdings wird dieses Format derzeit erst von wenigen Banken unterstützt. Insbesondere die großen Banken unterstützen dieses Format. Fragen sie daher vor dessen Einsatz bei der Bank nach.

Dazu sind drei Funktionen erforderlich, die nachfolgend dargestellt sind. Der logische Ablauf stellt sich folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateCGI | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_WriteCGI | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages übergeben. |
| … | |
| - Aufruf von SepaTools_CloseCGI | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |

Innerhalb dieser Funktionen finden umfangreiche Plausibilitätsprüfungen statt.

Die Möglichkeiten dieses erweiterten XML-Formats gehen noch weit über die hier dargestellte Implementierung hinaus. Validiert wurde diese Implementierung mit der NRODEA Bank in Finnland. Um u.U. notwendige Erweiterungen möglich zu machen, wurden großzügige Reservebereiche eingefügt.

# 85 SepaTools_CreateCGI

## *85.1 Zweck der Funktion*

Mit dieser Funktion wird die Ausgabe von XML-Dateien initialisiert. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

## *85.2 Funktionsaufruf*

**int SepaTools_CreateCGI( CGICreateStruct *CreateStruct)**

## *85.3 Parameter*

***CreateStruct*** Hier wird ein Pointer auf die Struktur CreateStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

struct CGICreateStruct

| | | | |
|---|---|---|---|
| { | char | ExportPfad[255+1] | Laufwerk und Pfad für die Ausgabe der XML-Datei. |
| | int | Lastschrift | Überweisung/Lastschrift. Wird derzeit ignoriert, nur Überw. |
| | char | XMLName[35+1] | Der Name der zu erzeugenden XML-Datei [1)]. |
| | char | MsgId[35+1] | Message-Id (Kennung) für die XML-Datei [2)]. |
| | char | EinreicherName[70+1] | Name des Einreichers der Datei (mind. 3 Zeichen). |
| | char | EinreicherId[35+1] | Optionale Einreicher-Id [3)] |
| | char | EinreicherSchema[4+1] | Optionales Einreicherschema, z.B. CUST [4)] |
| | int | ShortMsgId | Kurze EndToEnd-Id verwenden [5)] |
| | int | WhgSort | Sortierung nach Währung [6)] |
| | char | Reserve[996] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Unabhängig von der übergebenen oder nicht übergebenen Dateinamenserweiterung wird die Dateinamenserweiterung immer auf .xml gesetzt.

   Wird hier ein Leerstring oder der Wert Dummy.xml übergeben, so verwaltet die API den Dateienamen intern. Dies ist erforderlich, wenn die XML-Datei als Byte-Array im Speicher ausgegeben werden soll.

2) Die Message-Id soll eine möglichst eindeutige Identifizierung der XML-Datei sein. Wenn Sie hier einen Leerstring übergeben, so wird intern automatisch eine Message-Id gebildet und in der Struktur zurückgegeben.

3) Neben dem Namen des Einreicher der Zahlungen kann von der dateiannehmenden Bank eine Einreicher-Id zur internen Identifizierung des Einreichers verlangt werden.

   Diese Id wird von der Bank mitgeteilt.

4) Neben der optionalen Einreicher-Id wird von der Bank auch ein Schema-Name für die Einreicher-Id vorgegeben, z.B. CUST.

5) Lt. Regelwerk kann die EndToEnd-Id bis zu 35 Zeichen lang sein. Manche Banken verlangen aber eine max. 16stellige EndToEnd-Id.

   Wird der Wert dieses Feldes auf „1“ gesetzt, so wird eine eindeutige, 16stellige, rein numerische EndToEnd-Id gebildet.

6) Normalerweise werden die Daten unterschiedlicher Währungen gemischt in die XML-Datei geschrieben.

   Wenn Sie in diesem Feld dagegen den Wert **1** übergeben, so werden diese Datensätze auch nach der Währung sortiert. Wenn sich nun die Währung innerhalb der Daten ändert, wird innerhalb der XML-Datei ein neuer Sammler (Batch) gebildet.

   Ob dies erforderlich ist, ist mit der dateiempfangenden Bank zu klären.

### *85.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Es wurde ein formal unrichtiger Pfad (Länge < 2 Zeichen) für die Exportdatei übergeben. |
| -2 | Das angegebene Verzeichnis für die Exportdatei existiert nicht. |
| -3 | Der Datenträger, der im Pfadnamen der Exportdatei angegeben wurde ist schreibgeschützt. Die Datei kann nicht geschrieben werden. |
| -4 | Der Name des Einreichers der Daten wurde zu kurz (< 3 Zeichen) angegeben. |
| -5 | Die zum Ablauf der Funktion erforderliche temporäre Datei konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (TempPath). |
| -6 | Der Pfadname für die Exportdatei ist zu lange (>200 Zeichen). |
| -7 | Das angegebene Laufwerk im Pfad für die Exportdatei ist nicht bereit (möglicherweise kein Datenträger eingelegt). |
| -8 | Der Name für die XML-Datei wurde zu kurz (< 3 Zeichen) angegeben. |
| -9 | Die Funktion SepaTools_CreateCGI wurde bereits aufgerufen. Vor einem weiteren Aufruf muss zuvor die Funktion SepaTools_CloseCGI aufgerufen werden. |

| | |
|---|---|
| | Die Funktionalitäten zur Erzeugung einer XML-Datei sind nicht „Multi-Tread“ fähig! |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 86 SepaTools_WriteCGI

## *86.1 Zweck der Funktion*

Über jeden Aufruf dieser Funktion wird genau ein Datensatz für einen Zahlungsauftrag übergeben. Es werden umfangreiche Plausibilitätsprüfungen durchgeführt. Aufgrund des großen möglichen Umfangs dieser Implementierung, können die Plausiblitätsprüfungen nicht so streng durchgeführt werden wie bei normalen SEPA-Dateien.

Es können gemischt Zahlungsverkehrsaufträge von unterschiedlichen Auftraggebern übergeben werden. Die Aufträge werden automatisch nach Auftraggebern sortiert, so dass bei mehreren Zahlungsverkehrsaufträgen des gleichen Auftraggebers intern automatisch Sammler gebildet werden.

## *86.2 Funktionsaufruf*

**int SepaTools_WriteCGI(CGIWriteStruct *WriteStruct)**

## *86.3 Parameter*

***WriteStruct*** Hier wird ein Pointer auf die Struktur WriteStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

**Bankfachlicher Hinweis:**

Der Auftraggeber einer Zahlung ist immer der, der die Zahlung aktiv veranlasst. Bei Überweisungen ist dies der Zahler und bei Lastschriften ist dies der Zahlungsempfänger.

Der Empfänger der Zahlung ist damit der, der die Zahlung erhält. Bei Überweisungen ist dies der Begünstigte und bei Lastschriften ist dies der Zahlungspflichtige.

Derzeit sind aber nur Überweisungen möglich!

```
struct    CGIWriteStruct

{   int      KontrollSumen            Kontrollsummen (Betrag und Anzahl) anlegen [1)]
    char     PmInfoId[10+1]           PaymentInfoId – Kennzeichnung des Auftrags [2)].
    char     AusfDatum[8+1]           Ausführungsdatum der Zahlungen im Format TTMMJJJJ
    char     AuftragName[70+1]        Name des Auftraggebers der Zahlungen
    char     AuftragNameAbw[70+1]     Abweichender Name des Auftraggbers der Zahlungen
    char     AuftragLand[2+1]         ISO-Länderkennung des Landes des Auftraggebers z.B. US
    char     AuftragAdrLine1_Str[35+1]   Adresszeile 1der Adresse des Auftraggebers [3)]
    char     AuftragAdrLine2_Ort[35+1]   Adresszeile 2 der Adresse des Auftraggebres [3)]
    char     AuftragAdrHausNr[10+1]   Optional die Hausnummer der Adresse [3)]
    char     AuftragAdrPLZ[15+1]      Optional die Postleitzahl der Adresse, z.B. NY 10023 [3)]
```

| | | |
|---|---|---|
| int | AuftragAdrStruct | Unstrukt. (Wert=0) oder strukturierte (Wert=1) Adresse [3)] |
| char | AuftragId[35+1] | Optionale Auftrags-Id für den Zahlungsauftrag [4)] |
| char | AuftragSchema[4+1] | Optionales Schema für die Auftrags-Id [5)] |
| char | AuftragIBAN[35+1] | IBAN des Auftragskontos |
| char | AuftragBIC[11+1] | BIC des Auftragkontos |
| char | AuftragWhg[3+1] | ISO-Währungscode des Auftragskontos, z.B. USD |
| char | EndToEndId[10+1] | Kennung der Einzelzahlung [6)] |
| char | InstructionId[35+1] | Optionale Instruction-Id |
| int | BatchBooking | Batch-Booking Optionen [7)] |
| char | CatPurpose[4+1] | Optionaler Category Purpose Code |
| char | Betrag[12+1] | Betrag der Zahlung als Zeichenkette in Cent. |
| char | Whg[3+1] | Währung des Auftrags |
| char | EmpfName[70+1] | Name des Empfängers der Zahlung |
| char | EmpfNameAbw[70+1] | Abweichender Name des Empfängers der Zahlung |
| char | EmpfLand[2+1] | ISO-Ländercode des Empfängers der Zahlung |
| char | EmpfAdrLine1_Str[35+1] | Adresszeile 1der Adresse des Zahlungsempfängers [3)] |
| char | EmpfAdrLine2_Ort[35+1] | Adresszeile 2der Adresse des Zahlungsempfängers [3)] |
| char | EmpfAdrHausNr[10+1] | Optional die Hausnummer der Adresse [3)] |
| char | EmpfAdrPLZ[15+1] | Optional die Postleitzahl der Adresse, z.B. NY 10023 [3)] |
| int | EmpfAdrStruct | Unstrukt. (Wert=0) oder strukturierte (Wert=1) Adresse [3)] |
| char | EmpfIBAN[35+1] | IBAN oder Kontonummer des Zahlungsempfängers [8)] |
| char | EmpfKontoCode[4+1] | Schema des Kontos [12)] |
| char | EmpfBIC[11+1] | BIC des Empfängers [9)] |
| char | BankName[70+1] | Optional Name der Bank |
| char | BankLand[2+1] | ISO-Ländercode der Bank des Zahlungsempfängers |
| char | BankAdrLine1_Str[35+1] | Adresszeile 1der Adresse der Bank des Zahlungsempf. [3)] |
| char | BankAdrLine2_Ort[35+1] | Adresszeile 2der Adresse der Bank des Zahlungsempf. [3)] |
| char | BankAdrHausNr[10+1] | Optional die Hausnummer der Adresse [3)] |
| char | BankAdrPLZ[15+1] | Optional die Postleitzahl der Adresse, z.B. NY 10023 [3)] |
| int | BankAdrStruct | Unstrukt. (Wert=0) oder strukturierte (Wert=1) Adresse [3)] |
| char | BankClearingSysId[5+1] | Optional eine Bank Clearing Sys-Id [10)] |
| char | BankMemberId[35]+1 | Bank Member-Id zur Addressierung der Bank [9)] |
| char | Purpose[4+1] | Optionaler Purpose-Code |
| char | Zweck1[70+1] | Erster Teil des Verwendungszwecks [11)] |
| char | Zweck2[70+1] | Zweiter Teil des Verwendungszwecks [11)] |
| char | StructZweck[35+1] | Referenznummer, z.B. RF81123453 [11)] |
| char | StructTyp[4+1] | Typ der Referenz, z.B. SCOR [11)] |
| char | StructIssr[35+1] | Optionaler Herausgeber, z.B. ISO [11)] |
| int | DoStruct | Strukturieren Verwendungszweck anwenden (Wert=1) [11)] |
| char | Reserve[3000] | Reserve zur spätern Verwendung |

}

Zusätzliche Erklärungen:

1) Innerhalb der PaymentInfomation können Kontrollsummen für die Summe der Beträge und die Anzahl der Datensätze angelegt werden. Ob dies erforderlich oder/und zulässig ist, ist abhängig von der dateiempfangenden Bank.

   Setzen Sie den Wert dieses Feldes auf 1, wenn die Kontrollsummen geschrieben werden sollen.

2) Zur eindeutigen Kennzeichnung des Auftraggebers wird intern eine Kennung benötigt. Übergeben Sie hier ein bis zu 10stelliges Kürzel für diese Kennung. Intern wird das Kürzel dann um das Datum, die Uhrzeit und einer laufenden Nummer ergänzt.

Wenn Sie hier einen Leerstring übergeben, so wird die komplette Kennung automatisch gebildet.

Da aufgrund einer möglichen internen Sortierung die PmInfoId erst zu einem späteren Zeitpunkt im Rahmen des Aufrufs von SepaTools_CloseCGI gebildet wird, kann sie an dieser Stelle noch nicht über die Struktur zurückgeliefert werden. Das tatsächliche Ergebnis sehen Sie erst in der erzeugten XML-Datei.

3) Adressangaben können in zwei unterschiedlichen Varianten übergeben werden. Wenn eine Adressangabe in die XML-Datei geschrieben werden sollen, so ist zwingend das Feld „Land“ anzuliefern.

   Die Adresse kann entweder unstrukturiert in zwei Adresszeilen oder strukturiert mit den Werten Straße, Hausnummer, PLZ und Ort übergeben werden.

   Im Feld „*AdrLine1_Str“ steht daher entweder die Adresszeile 1 oder die Straße der Adresse. Im Feld „*AdrLine2_Ort“ steht dann entweder die Adresszeile 2 oder der Ort der Adresse.

   Ob die unstrukturierte oder die strukturierte Adresse verwendet wird, hängt von der Belegung des Feldes „*AdrStruct“ ab. Mit dem Wert 0 wird die unstrukturierte und mit Wert 1 die strukturierte Form verwendet.

4) Neben dem Namen des Auftraggebers der Zahlungen kann von der dateiannehmenden Bank optional eine Auftrags-Id zur internen Identifizierung des Auftraggebers verlangt werden.

   Diese Id wird von der Bank mitgeteilt.

5) Neben der optionalen Auftrags-Id wird von der Bank auch ein Schema-Name für die Auftrags-Id vorgegeben, z.B. BANK.

6) Zur eindeutigen Kennzeichnung des Zahlungsauftrages wird intern eine Kennung benötigt. Übergeben Sie hier ein bis zu 10stelliges Kürzel für diese Kennung ein. Intern wird das Kürzel dann um das Datum, die Uhrzeit und einer laufenden Nummer ergänzt.

   Wenn Sie hier einen Leerstring übergeben, so wird die komplette Kennung automatisch gebildet.

   Da aufgrund einer möglichen internen Sortierung die EndToEndId erst zu einem späteren Zeitpunkt im Rahmen des Aufrufs von SepaTools_CloseCGI gebildet wird, kann sie an dieser Stelle noch nicht über die Struktur zurückgeliefert werden. Das tatsächliche Ergebnis sehen Sie erst in der erzeugten XML-Datei.

7) Über den Wert BatchBooking wird entschieden, ab die einzelnen Buchungen beim Auftraggeber auf dem Kontoauszug als Einzelbuchungen oder als Sammelbuchungen erscheinen. Es gibt dabei folgende Ausprägungen:

   0 Die Buchung erfolgt entsprechend den Kontovorgaben der Bank
   1 Es erfolgen Einzelbuchungen, keine Sammelbuchung
   2 Es erfolgt eine Sammelbuchung

   Die Wirksamkeit dieses Wertes hängt aber von den Möglichkeiten der Bank ab. Es kann sein, dass diese Angaben sich nicht auswirken.

8) Hier wird die IBAN oder die Kontonummer des Zahlungsempfängers übergeben. Intern wird automatisch geprüft, ob es sich um eine IBAN oder eine Kontonummer handelt. Die richtige Behandlung in der XML-Datei erfolgt dann automatisch.

9) Die Adressierung der Bank des Zahlungsempfängers kann mittels eines BICs oder einer Clearing Member-Id erfolgen. Es darf daher nur einer dieser beiden Werte angeliefert werden.

   Für den Fall, dass der BIC ungültig ist, oder eine Member-Id angeliefert wird, hat die Clearing Member-Id Vorrang.

10) Im Zusammenhang mit Member-Id kann optional eine Bank Clearing Sys-Id (z.B. USABA) vergeben werden.

11 Der Verwendungszweck kann strukturiert (als Reffernznummer, z.B. RF54565465465465) oder unstrukturiert übergeben werden. Die beiden Zeilen des unstruktutrierten Verwendungszwecks werden in der XML-Datei zu einer Zeile zusammengesetzt.

   Das Feld „DoStruct“ gibt an, ob der Verwendungszweck strukturiert (Wert=1) oder unstrukturiert (Wert=0) übergeben werden soll.

12 Wird zur Adressierung des Zahlungsempfängers keine IBAN, sondern eine nationale Kontonummer verwendet, so muss u.U. ein Schema für dieses Konto angegeben werden.
Z.B. BBAN, beutet nationales Konto.

### *86.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Das Schreiben des Datensatzes in die temporäre Datei ist gescheitert. |
| -2 | Die Funktion SepaTools_CreateCGI wurde noch nicht aufgerufen. |
| -101 | Der Name des Zahlungsempfänger wurde nicht übergeben, oder ist kürzer als 3 Zeichen. |
| -104 | Es wurde kein Betrag übergeben, oder der Betrag ist kleiner 0 (negativ). |
| -153 | Es wurde keine IBAN für den Auftraggeber übergeben. |
| -163 | Das Ausführungsdatum ist ungültig, oder liegt in der Vergangenheit. |
| -401 | Es wurde keine Message-Id übergeben. |
| -402 | Es wurde kein BIC für den Auftraggeber übergeben. |
| -403 | Es wurde eine ungültige Währung für das Konto des Auftraggebers übergeben. |
| -404 | Es wurde keine EndToEnd-Id übergeben. |

| | |
|---|---|
| -405 | Es wurde eine ungültige Währung für die Zahlung übergeben. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 87 SepaTools_CloseCGI

## *87.1 Zweck der Funktion*

Mit dieser Funktion wird die eigentliche XML-Datei aus der von SepaTools_WriteCGI erzeugten temporären Datei erzeugt und in die definierte Datei geschrieben.

## *87.2 Funktionsaufruf*

**int SepaTools_CloseCGI( XMLCloseStruct *CloseStruct)**

## *87.3 Parameter*

***CloseStruct*** Hier wird ein Pointer auf die Struktur CloseStruct übergeben. Die Struktur ist nachfolgend dargestellt. Die Struktur wird leer übergeben und von der API mit Informationen für die Applikation gefüllt.

```
struct    XMLCloseStruct

{   int       Anzahl                Anzahl der erzeugten Datensätze.
    char      SummeBetrag[12+1]     Gesamtsumme der Beträge in der XML-Datei als
                                    Zeichenkette in Cent.
}
```

## *87.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Die XML-Datei konnte nicht erzeugt/initialisiert werden. Bitte setzen Sie sich mit dem Hersteller in Verbindung. |
| -2 | Beim Erzeugen der einzelnen Datensätze ist ein Fehler aufgetreten. Bitte setzen Sie sich mit dem Hersteller in Verbindung. |
| -3 | Beim Schreiben der XML-Datei auf den Datenträger ist ein Fehler aufgetreten. |

Die folgenden Returncodes -4 bis -7 können nur auftreten, wenn im „Speichermodus“ gearbeitet wird (es wurde kein Dateiname für die XML-Datei übergeben). Die XML-Datei wird nicht als Datei, sondern über die Funktion SepaTools_XMLGetData im Speicher übergeben.

| | |
|---|---|
| -4 | Es stehen keine Daten zur Ausgabe der XML-Datei im Speicher zur Verfügung. |
| -5 | Die Größe der XML-Daten ist größer als die maximal erlaubte Größe von derzeit 500 MB. |
| -6 | Der Speicherbereich zu Übergabe der Daten konnte nicht belegt werden. Möglicherweise steht zu wenig Hauptspeicher zur Verfügung. |
| -7 | Die Ausgabe der XML-Daten in den Speicherbreich ist fehlgeschlagen. |

| | |
|---|---|
| -8 | Es konnten keine Datensätze geschrieben werden, da alle Datensätze fehlerhaft waren. |
| -9 | Die temporäre Datei konnte nicht geöffnet werden. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 88 Neues AZV-Format (XML) ab 2022, zwingend ab 2025

Das DTAZV-Verfahren wird mittelfristig, ab dem November 2025 abgelöst. Ersetzt wird es durch das XML-Format pain.001.001.09 nach dem ISO-Standard 20022.

Ursprünglich war geplant, dass dieses Format parallel zum DTAZV-Format schon im November 2021 zum Einsatz kommt. Dieser Termin wurde jetzt auf den November 2022 verschoben(zumindest für Deutschland).

Ab November 2022 wird dieses Format dann von den deutschen Banken, bis November 2025 parallel zum bisherigen DTAZV-Format, unterstützt. Ab November 2025 wird das bisherige DTAZV-Format nicht mehr angeboten.

Das Format pain.001.001.09 wird von SepaTools unterstützt. Grundlage dafür ist das „Technische Validierungsset“ (TVS, die XSD-Datei) von ISO 20022.

Bei der Implementation dieses Formats wurde darauf geachtet, dass auch Elemente unterstützt werden, die von der Deutschen Kreditwirtschaft (DK) nicht empfohlen und damit auch nicht unterstützt werden. Dies eröffnet aber die Möglichkeit die Implementierung auch in anderen Ländern einzusetzen und Zahlungen dort einzureichen.

Wenn es Abweichungen zu der DK-Variante gibt, so wie in der Dokumentation darauf hingewiesen.

Da die Implementierung dieses Formats deutlich komplexer als bei normalen SEPA-Zahlungen ist, wurde die Spezifikation schon frühzeitig bereitgestellt.

## *88.1 Abgrenzung zum CGI-Format*

Auch mit dem CGI-Format können Auslandszahlungen außerhalb des SEPA-Raums und in unterschiedlichen Währungen durchgeführt werden. Das CGI-formt ist allerdings eine Kooperation von mehreren Großbanken und wird weit nicht von allen Banken unterstützt.

Das Format pain.001.001.09 wurde aber im EPC entwickelt und sollte für (fast) alle Länder gültig sein.

## *88.2 Bankfachliche Aspekte*

In einigen Feldern werden codierte Inhalte erwartet. Die Inhalte, die Sie selbst beinflussen können, sind in den nachfolgenden Tabellen dargestellt.

### 88.2.1 Service-Level

| ***Code*** | ***Bedeutung*** |
|---|---|
| NURG | Nicht eilige Zahlung |
| URGP | Eilige Zahlung |
| SDVA | Sonstiger, vereinbarter Service-Level |

### 88.2.2 Codes zum Tragen von Entgelten

| *Code* | *Bedeutung* |
|---|---|
| DEBT | Alle Entgelte werden vom Zahler (Auftraggeber) getragen, d.h. der volle Überweisungsbetrag kommt beim Empfänger an. |
| CRED | Alle Entgelte werden vom Zahlungsempfänger getragen, d.h. der Überweisungsbetrag kommt abzüglich aller Entgelte an. |
| SHAR | Entgeltteilung – Der Zahler zahlt die Entgelte seiner Bank und der Zahlungsempfänger die übrigen Entgelte, d.h. der Überweisungsbetrag kommt abzüglich der übrigen Entgelte an. |
| SLEV | Sonstiger, vereinbarter Service Level |

Bei Scheckzahlungen (d.h. PaymentMethod = `CHK`) ist nur `SHAR` zulässig.

### 88.2.3 Zulässige Scheck Typen

| *Code* | *Name* | *Bedeutung* |
|---|---|---|
| CCHQ | CustomerCheque | Scheck gezogen auf das Konto des Auftraggebers (Debitors). Das Konto des Debitors wird belastet, wenn der Scheck eingelöst wurde. |
| CCCH | CertifiedCustomer-Cheque | Scheck gezogen auf das Konto des Auftraggebers (Debitors). Das Konto des Debitors wird belastet, wenn der Scheck eingelöst wird.<br><br>Die die Scheckzahlung erhaltende Bank druckt den Scheck und zertifiziert ihn. Damit ist die Zahlung garantiert. |
| BCHQ | BankCheque | Scheck gezogen auf das Konto des Auftraggebers (Debitor). Das Konto wird belastet, wenn der Scheck ausgestellt wird.<br><br>Der Scheck wird von der scheckerhaltenden Bank gedruckt. Diese garantiert die Zahlung. |
| DRFT | Draft | Ein garantierter Bankscheck mit einer Zahlung in der Zukunft zu einem bestimmten Termin. |
| ELDR | ElectronicDraft | Ein Instrument mit einem zukünftigen Valutadatum (nicht zahlen bevor . . .).<br><br>Der Begünstigte kann eine vorzeitige Bezahlung unter Abzug von Diskont erhalten.<br><br>Das Konto des Scheckausstellers wir zum angegebenen Valutadatum belastet. |

### 88.2.4 Zulässige Scheck-Lieferarten

| *Code* | *Name* | *Bedeutung* |
|---|---|---|
| MLDB | MailToDebtor | Der Scheck wird mittels Post an den Debitor gesendet. |
| MLCD | MailToCreditor | Der Scheck wird mittels Post an den Begünstigten gesendet. |
| MLFA | MailToFinalAgent | Der Scheck wird mittels Post an das endbeteiligte Kreditinstitut gesendet. |
| CRDB | CurierToDebtor | Der Scheck wird mittels Kurier an den Debitor überbracht. |

| *Code* | *Name* | *Bedeutung* |
|---|---|---|
| CRCD | CurierToCreditor | Der Scheck wird mittels Kurier an den Begünstigten überbracht. |
| CRFA | CurierToFinalAgent | Der Scheck wird mittels Kurier an das endbeteiligte Kreditinstitut gesendet. |
| PUDB | PickUpByDebtor | Der Scheck wird vom Debitor abgeholt. |
| PUCD | PickUpByCreditor | Der Scheck wird vom Begünstigten abgeholt. |
| PUFA | PickUpByFinalAgent | Der Scheck wird vom endbegünstigten Kreditinstitut abgeholt. |
| RGDB | RegisteredMailToDebtor | Der Scheck wird per Einschreiben an den Debitor gesandt. |
| RGCD | RegisteredMailToCreditor | Der Scheck wird per Einschreiben an den Begünstigten gesandt. |
| RGFA | RegisteredMailToFinalAgent | Der Scheck wird per Einschreiben an das endbeteiligte Kreditinstitut gesandt. |

### 88.2.5 Weisungsschlüssel

| *Code* | *Bedeutung* | *Einschränkung* |
|---|---|---|
| CHQB | Nur mittels Scheck zahlen | Darf nicht mit Weisungsschlüssel `HOLD` kombiniert werden.<br><br>Nicht verwendbar im Falle <CtgyPurp> = `CORT` oder `INTC` |
| HOLD | Nur nach Identifikation zahlen | Darf nicht mit Weisungsschlüssel `CHQB` kombiniert werden.<br><br>Nicht verwendbar im Falle <CtgyPurp> = `CORT` oder `INTC` |
| PHOB | An Begünstigten per Telefon avisieren | Darf nicht mit Weisungsschlüssel `TELB` kombiniert werden. |
| TELB | An Begünstigten auf effektivste Weise per Telekommunikation avisieren | Darf nicht mit Weisungsschlüssel `PHOB` kombiniert werden. |

### 88.2.6 Codes für den strukturierten Verwendungszweck

| *Code* | *Name* | *Bedeutung* |
|---|---|---|
| RADM | RemittanceAdviceMessage | Es handelt sich um einen Hinweis auf eine Überweisung. |
| ROIN | RelatedPaymentInstruction | Es handelt sich um eine an Bedingungen verknüpfte Zahlung (z.B. Deckungsanforderung). |
| FXDR | ForeignExchangeDealReference | Es handelt sich um eine im Voraus vereinbarte Devisentransaktion. |
| DISP | DispatchAdvice | Das Dokument ist ein Versandhinweis. |
| PUOR | PurchaseOrder | Das Dokument ist eine Bestellung. |
| SCOR | StructuredCommunicationReference | Es handelt sich um eine strukturierte Information mit Bezug auf die tatsächliche Zahlung. |

# 89 AZV Neu Dateien erzeugen

Wenn es Abweichungen zu der DK-Variante gibt, so wie in der Dokumentation darauf hingewiesen.

Da die Implementierung dieses Formats deutlich komplexer als bei normalen SEPA-Zahlungen ist, wurde die Spezifikation schon frühzeitig bereitgestellt.

Für den Ablauf sind drei Funktionen erforderlich, die nachfolgend dargestellt sind. Der logische Ablauf stellt sich folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateAZVN | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_WriteAZVN | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages übergeben. |
| … | |
| - Aufruf von SepaTools_CloseAZVN | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |

Innerhalb dieser Funktionen finden umfangreiche Plausibilitätsprüfungen statt.

Um u.U. notwendige Erweiterungen möglich zu machen, wurden großzügige Reservebereiche eingefügt.

## *89.1 Abgrenzung zum CGI-Format*

Auch mit dem CGI-Format können Auslandszahlungen außerhalb des SEPA-Raums und in unterschiedlichen Wahrungen durchgeführt werden. Das CGI-formt ist allerdings eine Kooperation von mehreren Großbanken und wird weit nicht von allen Banken unterstützt.

Das Format pain.001.001.09 wurde aber im EPC entwickelt und sollte für (fast) alle Länder gültig sein.

# 90 Verwendung von Adressen

Innerhalb der Datenstrukturen werden häufiger Adressen (teilweise optional) verwendet. An dieser Stelle sei daher die grundsätzliche Verwendung der Adressen beschrieben.

Grundsätzlich sollte die strukturierte Angabe der Adresse Verwendet werden. Alternativ stehen zwei allgemeine Adresszeilen zur Verfügung. Damit ergäbe sich für den Adressatz folgende beispielhafte Struktur:

```
struct    AdrStruct

{   char    xxAdrLand[2+1]              Länderkennung der Adresse
    char    xxAdrLine1_Strasse[35+1]    Straße oder erste Adresszeile
    char    xxAdrLine2_Ort[35+1]        Ort oder zweite Adresszeile
    char    xxAdrHausnummer[10+1]       Hausnummer
    char    xxAdrPLZ[15+1]              Postleitzahl
    int     xxAdrStruct                 0=unstrukturierte, 1=strukturierte Adresse verwend.
    char    xxCtryOfRes[2+1]            Tatsächliche Residenz des Adressinhabers
}
```

Damit überhaupt eine Adresse geschrieben wird, muss zumindest das Feld xxAdrLand gültig befüllt werden. Die anderen Felder sind, zumindest aus Sicht der API optional, können aber bankfachlich gefordert sein.

Die Belegung des Feldes xxAdrStruct entscheidet, ob die strukturierte Adresse verwendet wird, oder nur die beiden Adresszeilen. Wird aber das Feld xxAdrPLZ gefüllt, so wird automatisch die strukturierte Adresse, unabhängig vom Feld xxAdrStruct verwendet.

**Hinweis:**

**Bitte geben Sie in Deutschland für den AZV immer die strukturierte Adresse an. Unstrukturierte Adressen werden abgelehnt.**

Das Feld xxCtryOfRes ist eigentlich kein Bestandteil der Adresse, sondern ein separates Feld. Es gibt das juristische Land an, in dem das Unternehmen ihren Sitz hat. Das Feld muss immer dann belegt sein, wenn das Feld xxAdrLand nicht belegt ist (d.h. die Adresse fehlt) oder wenn das Feld xxAdrLand vom Feld xxCtryOfRes abweicht.

Wird dieses Feld nicht belegt, obwohl es notwendig wäre, so wird beim Debitor und beim Creditor versucht das Länderkennzeichen aus dem BIC und alternativ aus der IBAN zu ermitteln. Gelingt dies nicht, so ergibt das einen Fehlercode.

Nicht bei allen Adressen, muss dieses Feld belegt werden. Siehe hierzu die Datenstrukturen. Innerhalb der Datenstrukturen wird damit auf die Interpretation der einzelnen Datumsfelder nicht mehr explizit eingegangen.

Länderkennungen werden immer auf Richtigkeit nach dem hinterlegten Länderverzeichnis geprüft. Ist eine Länderkennung vorhanden und diese ist falsch, so kann dieser Datensatz nicht geschrieben werden. Ein Fehlercode wird dann übergeben.

Eine Adresse wird als gültig betrachtet, wenn mindestens eine gültige Länderkennung für die Adresse angeliefert wurde.

**Achtung:**

Soweit möglich, sollte immer der strukturierte Verwendungszweck verwendet werden.

# 91 SepaTools_CreateAZVN

## *91.1 Zweck der Funktion*

Mit dieser Funktion wird die Ausgabe von XML-Dateien initialisiert. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

## *91.2 Funktionsaufruf*

**int SepaTools_CreateAZVN(AZVNCreateStruct *CreateStruct)**

## *91.3 Parameter*

***CreateStruct*** Hier wird ein Pointer auf die Struktur CreateStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

struct AZVNCreateStruct

```
{   char    ExportPfad[255+1]          Laufwerk und Pfad für die Ausgabe der XML-Datei.
    int     Lastschrift                Überweisung/Lastschrift. Wird derzeit ignoriert, nur Überw.
    char    XMLName[35+1]              Der Name der zu erzeugenden XML-Datei [1].
    char    XMLNameSepa[35+1]          Name der optionalen SEPA-Datei [7]
    int     TrySepa                    Wenn möglich auch SEPA-Datensätze erzeugen [7]
    char    MsgId[35+1]                Message-Id (Kennung) für die XML-Datei [2].
    char    EinreicherName[70+1]       Name des Einreichers der Datei (mind. 3 Zeichen).
    char    EinreicherId[35+1]         Optionale Einreicher-Id [3]
    char    EinreicherSchema[4+1]      Optionales Einreicherschema, z.B. CUST [4]
    char    EinreicherIssr[35+1]       Optionaler Herausgeber der EinreicherId [9]
    int     ShortMsgId                 Kurze EndToEnd-Id verwenden [5]
    int     EndToEndUpper              EndToEnd-Id in Großbuchstaben wandeln (Wert=1)
    int     WhgSort                    Sortierung nach Währung [6].
    char    Land[2+1]                  Optional das Land des Einreichers [8]
    char    AdrLandLine1_Strasse[35+1]      Optional die Strasse des Einreichers [8]
    char    AdrLandLine2_Ort[35+1]     Optional der Ort des Einreichers [8]
    char    Hausnummer[10+1]           Optional die Hausnummer des Einreichers [8]
    char    PLZ[15+1]                  Optional die Postleitzahl [8]
    int     DoStruct                   Flag für strukturierte Adresse [8]
    char    Reserve[1000]              Reserve zur späteren Verwendung
}
```

Zusätzliche Erklärungen:

1) Unabhängig von der übergebenen oder nicht übergebenen Dateinamenserweiterung wird die Dateinamenserweiterung immer auf .xml gesetzt.

   Wird hier ein Leerstring oder der Wert Dummy.xml übergeben, so verwaltet die API den Dateienamen intern. Dies ist erforderlich, wenn die XML-Datei als Byte-Array im Speicher ausgegeben werden soll.

2) Die Message-Id soll eine möglichst eindeutige Identifizierung der XML-Datei sein. Wenn Sie hier einen Leerstring übergeben, so wird intern automatisch eine Message-Id gebildet und in der Struktur zurückgegeben.

3) Neben dem Namen des Einreicher der Zahlungen kann von der dateiannehmenden Bank eine Einreicher-Id zur internen Identifizierung des Einreichers verlangt werden.

   Diese Id wird von der Bank mitgeteilt.

4) Neben der optionalen Einreicher-Id wird von der Bank optional auch ein Schema-Name für die Einreicher-Id vorgegeben, z.B. CUST.

5) Lt. Regelwerk kann die EndToEnd-Id bis zu 35 Zeichen lang sein. Manche Banken verlangen aber eine max. 16stellige EndToEnd-Id.

   Wird der Wert dieses Feldes auf „1“ gesetzt, so wird eine eindeutige, 16stellige, rein numerische EndToEnd-Id gebildet.

6) Normalerweise werden die Daten unterschiedlicher Währungen gemischt in die XML-Datei geschrieben.

   Wenn Sie in diesem Feld dagegen den Wert **1** übergeben, so werden diese Datensätze auch nach der Währung sortiert. Wenn sich nun die Währung innerhalb der Daten ändert, wird innerhalb der XML-Datei ein neuer Sammler (Batch) gebildet.

   Ob dies erforderlich ist, ist mit der dateiempfangenden Bank zu klären.

7) Wird der Wert des Feldes TrySepa mit 1 belegt, so werden die übergebenen Datensätze daraufhin überprüft, ob die Zahlung auch als SEPA-Zahlung ausgeführt werden könnte. Ist dies der Fall, dann wird eine zweite XML-Daei mit den SEPA-Zahlungen erzeugt.

   Den Namen für diese, optionale zweite XML-Datei geben Sie bitte im Feld XMLNameSepa an. Wird hier kein Dateiname übergeben, obwohl das Feld TrySepa mit dem Wert 1 belegt ist, so wird der Dateiename aus dem Inhalt des Feldes XMLName mit der Ergänzung _Sepa gebildet.

   Beispiel:

   XMLName = Test.xml, dann wird XMLNameSepa = Test_Sepa.xml.

8) Bei der Einreichung in Deutschland (Einreicherland Deutschland) sind diese Werte nicht erlaubt. Wenn Sie diese Werte trotzdem angeben, so werden diese, wenn Einleicherland Deutschland ist, automatich ausgefiltert.

9) Der Herausgeber der Einreicher-Id darf in Deutschlöand nicht angegeben werden. Der Wert bleibt trotzdem erhalten, da er der ISO 20022 entspricht.

### *91.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Es wurde ein formal unrichtiger Pfad (Länge < 2 Zeichen) für die Exportdatei übergeben. |
| -2 | Das angegebene Verzeichnis für die Exportdatei existiert nicht. |
| -3 | Der Datenträger, der im Pfadnamen der Exportdatei angegeben wurde ist schreibgeschützt. Die Datei kann nicht geschrieben werden. |
| -4 | Der Name des Einreichers der Daten wurde zu kurz (< 3 Zeichen) angegeben. |
| -5 | Die zum Ablauf der Funktion erforderliche temporäre Datei konnte nicht geöffnet werden. Bitte überprüfen Sie den in der Funktion SepaTools_Init übergebenen Pfad (TempPath). |
| -6 | Der Pfadname für die Exportdatei ist zu lange (>200 Zeichen). |
| -7 | Das angegebene Laufwerk im Pfad für die Exportdatei ist nicht bereit (möglicherweise kein Datenträger eingelegt). |
| -8 | Der Name für die XML-Datei wurde zu kurz (< 3 Zeichen) angegeben. |
| -9 | Die Funktion SepaTools_CreateAZVN wurde bereits aufgerufen. Vor einem weiteren Aufruf muss zuvor die Funktion SepaTools_CloseAZVN aufgerufen werden.<br><br>Die Funktionalitäten zur Erzeugung einer XML-Datei sind nicht „Multi-Tread“ fähig! |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 92 SepaTools_WriteAZVN

## *92.1 Zweck der Funktion*

Über jeden Aufruf dieser Funktion wird genau ein Datensatz für einen Zahlungsauftrag übergeben. Es werden umfangreiche Plausibilitätsprüfungen durchgeführt. Aufgrund des großen möglichen Umfangs dieser Implementierung, können die Plausiblitätsprüfungen nicht so streng durchgeführt werden wie bei normalen SEPA-Dateien.

Es können gemischt Zahlungsverkehrsaufträge von unterschiedlichen Auftraggebern übergeben werden. Die Aufträge werden automatisch nach Auftraggebern sortiert, so dass bei mehreren Zahlungsverkehrsaufträgen des gleichen Auftraggebers intern automatisch Sammler gebildet werden.

## *92.2 Funktionsaufruf*

**int SepaTools_WriteAZVN(AZVNWriteStruct *WriteStruct)**

## *92.3 Parameter*

***WriteStruct*** Hier wird ein Pointer auf die Struktur WriteStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bei allen Zeichenketten wird intern geprüft, ob es sich um einen gültigen Zeichensatz handelt. Ist dies nicht der Fall, so werden ungültige Zeichen gelöscht. Die korrigierte Zeichenkette wird dann in der Struktur zurückgegeben.

**Bankfachlicher Hinweis:**

Der Auftraggeber einer Zahlung ist immer der, der die Zahlung aktiv veranlasst. Bei Überweisungen ist dies der Zahler und bei Lastschriften ist dies der Zahlungsempfänger.

Der Empfänger der Zahlung ist damit der, der die Zahlung erhält. Bei Überweisungen ist dies der Begünstigte und bei Lastschriften ist dies der Zahlungspflichtige.

Derzeit sind aber nur Überweisungen möglich!

```
struct    AZVNWriteStruct

{   int      KontrollSumen              Kontrollsummen (Betrag und Anzahl) anlegen [1)]
    char     PmInfoId[10+1]             PaymentInfoId – Kennzeichnung des Auftrags [2)].
    char     ExtPmInfoId[35+1]          Komplette Payment Info Id [23)]
    char     PayMethode[3+1]            Art der Zahlung TRF oder CHK [3)]
    char     AusfDatum[8+1]             Ausführungsdatum der Zahlungen im Format TTMMJJJJ
    char     AuftragName[70+1]          Name des Auftraggebers der Zahlungen
    char     AuftragLand[2+1]           ISO-Länderkennung des Landes des Auftraggebers z.B. US
    char     AuftragAdrLine1_Str[35+1]  Optional die Straße des Auftraggebers
    char     AuftragAdrLine2_Ort[35+1]  Optional der Ort des Auftraggebers
    char     AuftragHausnummer[10+1]    Optional die Hausnummer des Auftraggebers
```

| | | |
|---|---|---|
| char | AuftragPLZ[15+1] | Optional die Hausnummer des Auftraggebers |
| int | AuftragAdrStruct | Flag für strukturiere Adresse |
| char | AuftragCtryOfRes[2+1] | Residenz des Auftraggebers |
| char | AuftragAbwName[70+1] | Abweichender Name des Auftraggebers der Zahlungen [4)] |
| char | AuftragAbwLand[2+1] | Optional Land des abweichenden Auftraggebers [4)] |
| char | AuftragAbwAdrLine1_Str[35+1] | Optional die Strasse des abw. Auftraggebers [4)] |
| char | AuftragAbwAdrLine2_Ort[35+1] | Optional der Ort des abw. Auftraggebers [4)] |
| char | AuftragAbwAdrHausnummer[10+1] | Optional der Ort des abw. Auftraggebers [4)] |
| char | AuftragAbwAdrPLZ[15+1] | Optional die Postleitzahl des abw. Auftraggebers [4)] |
| int | AuftragAbwAdrStruct | Flag für die strukturierte Adresse |
| char | AuftragCtrOfRes[2+1] | Residenz des abweichenden Auftraggebers [4)] |
| int | AuftragAbwInTx | Abw. Auftraggeber in Transaktionsblock schreiben [5)] |
| char | AuftragId[35+1] | Optionale Auftraggeber-Id [6)] |
| char | AuftragSchema[4+1] | Optionales Schema für die AuftragId, z.B. BANK [7)] |
| char | AuftragIssr[35+1] | Optionaler Herausgeber der AuftragsId |
| char | AuftragIBAN[35+1] | Die IBAN des Auftraggebers |
| char | AuftragBIC[11+1] | Der BIC der Bank des Auftraggebers |
| char | AuftragKontoId[35+1] | Alternative Konto-Id des Auftraggebers [8)] |
| char | AuftragKontoCode[4+1] | Optionales Schema für die Konto-Id des Auftraggebers [8)] |
| char | AuftragKontoIssr[35+1] | Optionaler Herrausgeber für die Konto-Id des Auftragg. [8)] |
| char | AuftragWhg[3+1] | Die Währung des Auftrgkontos, ISO-Währungscode |
| char | EndToEndId[10+1] | Ein Kürzel zur Bildung der EndToEnd-Id [9)] |
| char | ExtEndToEndId[35+1] | Erweiterte EndToEnd-Id [9)] |
| char | InstructionId[35+1] | Optionale Instruction-Id für diese Zahlung |
| int | BatchBooking | Batch-Booking Optionen [10)] |
| int | UETR | Eindeutige UUID gemäß RFC 4122 generieren [11)] |
| char | ServiceLevel[4+1] | Service Level für diese Zahlung [12)] |
| char | CatPurpose[4+1] | Optionaler 4stelliger Category Purpose Code, z.B. SALA |
| char | Betrag[12+1] | Betrag der Zahlung in Cent |
| char | Whg[3+1] | Währung der Zahlung |
| char | CCyOfTrf[3+1] | Äquvalenzbetrag der Zahlung [13)] |
| char | Gebuehr[4+1] | Die Gebührenregelung für diese Zahlung [14)] |
| char | EmpfName[70+1] | Name des Empfängers der Zahlungen |
| char | EmpfLand[2+1] | ISO-Länderkennung des Landes des Empfängers z.B. US |
| char | EmpfAdrLine1_Str[35+1] | Optional die Straße des Empfängers |
| char | EmpfAdrLine2_Ort[35+1] | Optional der Ort des Empfängers |
| char | EmpfAdrHausnummer[10+1] | Optional die Hausnummer des Empfängers |
| char | EmpfAdrPLZ[15+1] | Optional die PLZ des Empfängers der Zahlung |
| int | EmpfAdrStruct | Flag für strukturiere Adresse |
| char | EmpfCtryOfRes[2+1] | Residenz des Empfängers der Zahlung |
| char | EmpfAbwName[70+1] | Name des abweichenden Empfängers der Zahlungen [15)] |
| char | EmpfAbwLand[2+1] | ISO-Länderkennung des Landes des abw. Empfängers |
| char | EmpfAbwAdrLine1_Str[35+1] | Optional die Straße des abw. Empfängers |
| char | EmpfAbwAdrLine2_Ort[35+1] | Optional der Ort des abw. Empfängers |
| char | EmpfAbwAdrHausnummer[10+1] | Optional die Hausnummer des abw. Empfängers |
| char | EmpfAbwAdrPLZ[15+1] | Optional die Hausnummer des abw. Empfängers |
| int | EmpfAbwAdrStruct | Flag für strukturiere Adresse |
| char | EmpfAbwCtryOfRes[2+1] | Residenz des abw. Empfängers der Zahlung |
| char | EmpfIBAN[35+1] | IBAN des Zahlungsempfängers [16)] |
| char | EmpfKontoCode[4+1] | Konto Code für ein alternatives Konto [16)] |
| char | EmpfBIC[11+1] | BIC der Bank des Empfängers [17)] |
| char | EmpfZwischenBank[11+1] | Durchleitung über eine Zwischenbank |
| char | BankName[70+1] | Optionaler Name der Bank des Zahlungsempfängers |
| char | BankLand[2+1] | ISO-Länderkennung des Landes der Empfängerbank |

| | | |
|---|---|---|
| char | BankAdrLine1_Str[35+1] | Optional die Straße der Bank |
| char | BankAdrLine2_Ort[35+1] | Optional der Ort der Bank |
| char | BankAdrHausnummer[10+1] | Optional die Hausnummer der Bank |
| char | BankAdrPLZ[15+1] | Optional die PLZ der Bank des Empfängers |
| int | BankAdrStruct | Flag für strukturiere Adresse |
| char | BankClearingSysId[5+1] | Optional die Clearing Sys-Id der Bank [17)] |
| char | BankMemberId[35+1] | Optional die Member-Id der Bank [17)] |
| char | ChkTyp[4+1] | Bei Scheckzahlungen, Typ des Schecks [18)] |
| char | ChkLieferung[4+1] | Art der Scheck Lieferung [19)] |
| char | ChkNummer[35+1] | Optional die Schecknummer |
| char | ChkEmpfName[70+1] | Optional der Name des Scheckempfängers |
| char | ChkEmpfLand[2+1] | ISO-Länderkennung des Landes des Scheckempfängers |
| char | ChkEmpfAdrLine1_Str[35+1] | Optional die Straße des Scheckempfängers |
| char | ChkEmpfAdrLine2_Ort[35+1] | Optional der Ort des Scheckempfängers |
| char | ChkEmpfAdrHausnummer[10+1] | Optional die Hausnummer des Scheckempfängers |
| char | ChkEmpfADrPLZ[15+1] | Optional die PLZ des Scheckempfängers |
| int | ChkEmpfAdrStruct | Flag für strukturiere Adresse |
| char | Weisung1_Code[4+1] | Weisung 1codiert [20)] |
| char | Weisung2_Code[4+1] | Weisung 2codiert [20)] |
| char | Weisung3_Code[4+1] | Weisung 3codiert [20)] |
| char | Weisung4_Code[4+1] | Weisung 4codiert [20)] |
| char | Weisung1_Text[35+1] | Weisung 1in Textform [20)] |
| char | Weisung2_Text[35+1] | Weisung 2in Textform [20)] |
| char | Weisung3_Text[35+1] | Weisung 3in Textform [20)] |
| char | Weisung4_Text[35+1] | Weisung 4in Textform [20)] |
| char | Purpose[4+1] | Optionaler Purpose Code für diese Zahlung, z.B. SALA |
| char | Zweck1[70+1] | Zeile 1 des unstrukturierten Verwendungszwecks [21)] |
| char | Zweck2[70+1] | Zeile 2 des unstrukturierten Verwendungszwecks [21)] |
| char | StructZweck[35+1] | Strukturierter Verwendungszweck in Textform [22)] |
| char | StructType[4+1] | Schema des strukt. Verwendungszwecks, z.B. SCOR [22)] |
| char | StructIssr[35+1] | Optional der Herrausgeber des Schemas [22)] |
| int | DoStructStruct | Flag für den strukturierten Verwendungszweck [22)] |
| char | Reserve[3000] | Reserve zur spätern Verwendung |

}

Zusätzliche Erklärungen:

1) Innerhalb der PaymentInfomation können Kontrollsummen für die Summe der Beträge und die Anzahl der Datensätze angelegt werden. Ob dies erforderlich oder/und zulässig ist, ist abhängig von der dateiempfangenden Bank.

   Setzen Sie den Wert dieses Feldes auf 1, wenn die Kontrollsummen geschrieben werden sollen.

2) Zur eindeutigen Kennzeichnung des Auftraggebers wird intern eine Kennung benötigt. Übergeben Sie hier ein bis zu 10stelliges Kürzel für diese Kennung. Intern wird das Kürzel dann um das Datum, die Uhrzeit und einer laufenden Nummer ergänzt.

   Wenn Sie hier einen Leerstring übergeben, so wird die komplette Kennung automatisch gebildet.

   Da aufgrund einer möglichen internen Sortierung die PmInfoId erst zu einem späteren Zeitpunkt im Rahmen des Aufrufs von SepaTools_CloseAZVN gebildet wird, kann sie an dieser

Stelle noch nicht über die Struktur zurückgeliefert werden. Das tatsächliche Ergebnis sehen Sie erst in der erzeugten XML-Datei.

Alternativ dazu können Sie im Feld ExtPmInfoId eine kompette Payment-Info-Id vergeben. Sie sind dann aber auch selbst dafür verantwortlich, dass diese Id für jeden Payment-Info Block innerhalb der ganzen Datei eindeutig ist.

Wenn Sie im Rahmen eines Verarbeitungslaufs unterschiedliche Payment-Ids verwenden, so wird für jede verwendete Payment-Id ein eigener Payment Information Block gebildet.

3) Geben Sie bitte hier an, ob es sich um Überweisungen (TRF) oder um Scheckzahlungen (CHK) handelt. Wird kein Wert übergeben, so wird das Feld intern auf den Wert TRF gesetzt.

Ein ungültiger Wert generiert einen negativen Returncode.

4) **DK-Regel:** Wenn ein abweichender Auftraggeber angegeben ist, so muss auch eine Adresse angegeben werden. Wird eine Adresse angegeben, so muss auch der Name des abweichenden Auftraggebers angegeben werden.

Der Name der Stadt des abweichenden Zahlungsempfängers ist immer anzugeben.

5) Die Daten des Auftraggebers und damit auch des abweichenden Auftraggebers stehen normalerweise im Payment Information Block. Damit werden aber die Daten des abweichenden Auftraggebers u.U. nicht bis zum Zahlungsempfänger weiter gegeben.

Wenn Sie dieses Feld mit dem Wert 1 belegen, so werden die Daten des abweichenden Auftraggebers anstatt in den Payment Information Bloch in den Transaktionblock geschrieben und damit sicher an den Zahlungsempfänger weiter gegeben.

6) Neben dem Namen des Auftraggebers der Zahlungen kann von der dateiannehmenden Bank optional eine Auftrags-Id zur internen Identifizierung des Auftraggebers verlangt werden.

Diese Id wird von der Bank mitgeteilt.

7) Neben der optionalen Auftrags-Id wird von der Bank optional auch ein Schema-Name für die Auftrags-Id vorgegeben, z.B. BANK.

8) Für den Auftraggeber sollte immer eine IBAN übergeben werden. Wenn das Auftraggeberkonto aber keine IBAN hat, so kann alternativ eine Konto-Id (z.B. Kontonummer) übergeben werden.

In der Regel wird dann von der Bank auch ein Schema-Name für die Konto-Id vergeben. Der Schem-Name ist immer 4stellig.

Optional kann in diesem Zusammenhang auch noch der Herausgeber dieser Id (z.B. Bankname) angegeben werden.

9) Die EndToEnd-Id für die einzelnen Zahlungen wird intern automatisch gebildet. Zur Individualisierung der zu erzeugenden Kennung, kann hier ein Kürzel mit bis zu 10 Zeichen angegeben werden.

Alternativ dazu können Sie im Feld ExtEndToEndId eine kompette EndToEnd-Id vergeben. Sie sind dann aber auch selbst dafür verantwortlich, dass diese Id innerhalb der ganzen Datei eindeutig ist.

10) Über den Wert BatchBooking wird entschieden, ab die einzelnen Buchungen beim Auftraggeber auf dem Kontoauszug als Einzelbuchungen oder als Sammelbuchungen erscheinen. Es gibt dabei folgende Ausprägungen:

    0 Die Buchung erfolgt entsprechend den Kontovorgaben der Bank
    1 Es erfolgen Einzelbuchungen, keine Sammelbuchung
    2 Es erfolgt eine Sammelbuchung

    Die Wirksamkeit dieses Wertes hängt aber von den Möglichkeiten der Bank ab. Es kann sein, dass diese Angaben sich nicht auswirken.

11) In Absprache mit der Bank kann es erforderlich sein, dass für jeden Datensatz eine eindeutige Id gemäß RFC 4122 gebildet werden muss. Wenn das erforderlich ist, so belegen Sie dieses Feld bitte mit dem Wert 1.

12) Es sind nur die Werte NURG (Non-Urgent Payment), URGP (Urgent Payment) und SDVA (Same Day Value) zugelassen. Für taggleiche Eilüberweisungen in Euro darf nur URGP verwendet werden.

    Im Falle von Scheckzahlungen (d.h. PaymentMethod CHK, s.o.) ist nur NURG zulässig.

    Wenn in diesem Feld keine Wert übergeben wird, so wird das Feld intern mit dem Wert NURG gefüllt.

13) Diese Währung ist die Währung, mit der die Zahlung auf die Reise geschickt wird. Es gibt aber Fälle, in denen die Zahlung beim Empfänger in eine andere Währung konvertiert werden soll.

    Wird z.B. das Feld Währung mit EUR belegt und das Feld CcyOfTrf mit USD, so wird der angegebene Betrag in Euro beim Empfänger in US-Dollar konvertiert.

14) Es sind folgende Werte lt. Tabelle erlaubt:

    Wird kein Service Level angegeben, so wird intern der Wert SHAR eingesetzt.

15) **DK-Regel:** Wenn ein abweichender Zahlungsempfänger angegeben ist, so muss auch eine Adresse angegeben werden. Wird eine Adresse angegeben, so muss auch der Name des abweichenden Zahlungsempfängers angegeben werden.

    Der Name der Stadt des abweichenden Zahlungsempfängers ist immer anzugeben.

16) Übergeben Sie bitte hier eine IBAN, oder eine andere Kontnummer des Zahlungsempfängers. Es gibt Zielländer (z.B. die USA oder China), die keine IBAN haben.

    Wenn die IBAN gültig ist, so wird diese verwendet. Ansonsten wird der Wert dieses Feldes als Kontonummer oder ein anderes Identikfikationsmerkmal des Kontos interpretiert.

    Im Falle einer Kontonummer können Sie optional einen Code für das Konto übergeben. Z.B. BBAN (Basic Bank Account Number).

17) Im Idealfall geben Sie hier den BIC der Bank des Zahlungsempfängers an. Es gibt aber auch Banken ohne BIC. In diesem Fall können Sie die Bank Clearing System-Id der Bank gemeinsam mit der Member-Id angeben (teilweise in den USA).

    Wird eine Member-Id angegeben, so wird anstelle des BICs die Bank Clearing System-Id und die Member-Id verwendet.

18) Bei Scheckzahlungen geben Sie hier bitte den Typ des Schecks an. Erlaubt sind die Werte lt. Tabelle:

    Wird kein Typ für den Scheck übergeben, so wird intern der Typ CCHQ (Kundenscheck) eingesetzt.

19) Geben Sie hier bitte an, wie der Scheck an den Empfänger geliefert werden soll. Dabei sind folgende Werte lt. Tabelle erlaubt:

    Wird kein Wert für die Lieferung übergeben, so wird intern der Wert CRDC eingesetzt.

20) Zu einer Zahlung können bis zu 4 Weisungen erteilt werden. Die Weisungen werden in codierter Form angegeben. Dabei sind folgende Werte lt. Tabelle gültig:

    Optional können zu den einzelnen Weisungsschlüsseln noch Weisungen in Textform dazu kommen.

    **DK-Regel:** In Deutschland sind nur maximal zwei Weisungen erlaubt.

21) Diese beiden Zeilen dienen dem unstrukturierten Verwendungszweck. Sie werden intern zu dem max. 140stelligen Verwendungszweck zusammengesetzt.

22) Im strukturiertem Verwendungszweck wird typischerweise eine Referenznummer übergeben. Über das Feld DoStruct wird entschieden, ob der strukturierte oder der unstrukturierte Verwendungszweck zu Anwendung kommt.

    Im Feld StructTyp sind nur die Werte DISP, FXDR, PUOR, RADM, RPIN und SCOR erlaubt.

    Beim strukturierten Verwendungszweck gibt es noch eine Vielzahl von Unterscheidungsmöglichkeiten und Ergänzungen. Auf Grund der Komplexität dieser Möglichkeiten, wurde auf diese vorerst verzichtet.

    Hier ist die Praxis abzuwarten, welche Optionen hier noch sinnvoll sind.

23) Im Feld PmInfoId wird normalerweise ein bis zu 10stelliges Kürzel für die Bildung der Payment Info Id übergeben. Die eigentliche Id wird dann unter Verwendung dieses Kürzels automatsich ermittelt.

    Alternativ dazu kann aber im Feld ExtPmInfoId die komplette Payment Info Id übergeben werden.  Die Applikation ist dann aber selbt dafür verantwortlich, dass diese Id innerhalb der Datei eindeutig ist.

### *92.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich. |
| -1 | Das Schreiben des Datensatzes in die temporäre Datei ist gescheitert. |
| -2 | Die Funktion SepaTools_CreateAZVN wurde noch nicht aufgerufen. |
| -101 | Der Name des Zahlungsempfänger wurde nicht übergeben, oder ist kürzer als 3 Zeichen. |
| -104 | Es wurde kein Betrag übergeben, oder der Betrag ist kleiner 0 (negativ). |
| -151 | Es wurde kein Name für den Auftraggeber übergeben. |
| -162 | Es wurde kein Name für den Einreicher der Datei übergeben. |
| -163 | Das Ausführungsdatum ist ungültig, oder liegt in der Vergangenheit. |
| -401 | Es wurde keine Message-Id übergeben. |
| -402 | Es wurde kein BIC für den Auftraggeber übergeben. |
| -403 | Es wurde eine ungültige Währung für das Konto des Auftraggebers übergeben. |
| -404 | Es wurde keine EndToEnd-Id übergeben. |
| -405 | Es wurde eine ungültige Währung für die Zahlung übergeben. |
| -406 | Das Land für den Auftraggeber ist ungütig. |
| -407 | Das Land für den Empfänger ist ungültig. |
| -408 | Das Land für die Bank des Zahlungsempfängers ist ungültig. |
| -409 | Das Land für den abweichenden Auftraggeber ist ungütig. |
| -410 | Das Land für den abweichenden Empfänger ist ungültig. |
| -411 | Das Land für den Scheckempfänger ist ungültig. |
| -451 | Das Land für die Residenz des Auftraggebers fehlt. |
| -452 | Das Land für die Residenz des Auftraggebers ist ungültig. |
| -453 | Für das Auftraggeberkonto wurde weder eine IBAN, noch eine andere Id übergeben. |
| -454 | Für das Auftraggeberkonto wurde keine Währung angegeben. |
| -455 | Abw. Auftragg. und Adresse des abw. Auftragg. müssen immer gemeinsam auftreten, oder beide nicht. |
| -456 | Die Länderkennung für die Residenz des abweichenden Auftraggebers fehlt. |

-457 Die Länderkennung für die Residenz des abweichenden Auftraggebers ist ungültig.

-458 Falsche Service Angabe. Gültige Werte sind nur NURG, URGP und SDVA.

-459 Ungültige Zahlungsmethode. Gültige Werte sind nur TRF und CHK.

-460 Die Währung für die Konvertierung der Zahlung ist ungültig.

-461 Die Gebührenregelung ist ungültig.

-462 Die angegebene Weisung (nur CHQB, HOLD, PHOB und TELB erlaubt) ist ungültig.

-463 Abw. Empfänger und Adresse des abw. Empfängers müssen immer gemeinsam auftreten, oder beide nicht.

-464 Die Länderkennung für die Residenz des Empfängers fehlt.

-465 Die Länderkennung der Residenz des abweichenden Empfängers ist ungültig.

-466 Die Länderkennung für die Residenz des abweichenden Empfängers fehlt.

-467 Die Länderkennung der Residenz des abweichenden Empfängers ist ungültig.

-468 Der Scheck-Typ ist ungültig. Siehe bitte Dokumentation.

-469 Die Scheck-Zustellart ist ungültig. Siehe bitte Dokumentation.

-470 Der Name des Scheckempfängers und seine Adresse müssen gemeinsam, oder nicht angegeben werden.

-471 Die Länderkennung für die Adresse des Scheckempfängers fehlt.

-472 Die Länderkennung des Scheckempfängers ist ungültig.

-473 Der Purpose-Code muss zwingend 4stellig sein.

-474 Der Typ für den strukturieren Verwendungszweck ist ungültig. Siehe bitte Dokumentation.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 93 SepaTools_CloseAZVN

## *93.1 Zweck der Funktion*

Mit dieser Funktion wird die eigentliche XML-Datei aus der von SepaTools_WriteAZVN erzeugten temporären Datei erzeugt und in die definierte Datei geschrieben.

## *93.2 Funktionsaufruf*

**int SepaTools_CloseAZVN( AZVNCloseStruct *CloseStruct)**

## *93.3 Parameter*

***CloseStruct*** Hier wird ein Pointer auf die Struktur CloseStruct übergeben. Die Struktur ist nachfolgend dargestellt. Die Struktur wird leer übergeben und von der API mit Informationen für die Applikation gefüllt.

```
struct   AZVNCloseStruct

{   int      Anzahl              Anzahl der erzeugten Datensätze.
    char     SummeBetrag[12+1]   Gesamtsumme der Beträge in der XML-Datei als
    int      AnzahlSepa          Anzahl der erzeugten SEPA-Datensätze.
    char     SummeBetragSepa[12+1] Gesamtsumme der Beträge in der optionalen
                                 SEPA-XML-Datei als Zeichenkette in Cent.
}
```

## *93.4 Returncodes*

Ist der Rückgabewert positiv, so ist dieser bitcodiert (OR-Verbindung) und hat dabei folgende Bedeutung:

1 Es konnten keine AZV-Datensätze geschrieben werden, da alle Datensätze fehlerhaft waren, oder in den Daten keine AZV-Daten enthalten waren.

2 Es konnten keine SEPA-Datensätze geschrieben werden, da alle Datensätze fehlerhaft waren, oder in den Daten keine SEPA-Daten enthalten waren.

Ansonsten hat der Rückgabewert folgende Bedeutung:

0 Alles verlief erfolgreich.

-1 Die XML-Datei konnte nicht erzeugt/initialisiert werden. Bitte setzen Sie sich mit dem Hersteller in Verbindung.

-2 Beim Erzeugen der einzelnen Datensätze ist ein Fehler aufgetreten. Bitte setzen Sie sich mit dem Hersteller in Verbindung.

-3 Beim Schreiben der XML-Datei auf den Datenträger ist ein Fehler aufgetreten.

Die folgenden Returncodes -4 bis -7 können nur auftreten, wenn im „Speichermodus“ gearbeitet wird (es wurde kein Dateiname für die XML-Datei übergeben). Die XML-Datei wird nicht als Datei, sondern über die Funktion SepaTools_XMLGetData im Speicher übergeben.

-4 Es stehen keine Daten zur Ausgabe der XML-Datei im Speicher zur Verfügung.

-5 Die Größe der XML-Daten ist größer als die maximal erlaubte Größe von derzeit 500 MB.

-6 Der Speicherbereich zu Übergabe der Daten konnte nicht belegt werden. Möglicherweise steht zu wenig Hauptspeicher zur Verfügung.

-7 Die Ausgabe der XML-Daten in den Speicherbreich ist fehlgeschlagen.

-8 Es konnten keine Datensätze für eine AZV-XML-Datei geschrieben werden, da alle Datensätze fehlerhaft waren.

-9 Die temporäre Datei konnte nicht geöffnet werden.

-10 Es konnten keine Datensätze für eine SEPA-XML-Datei geschrieben werden, da alle Datensätze fehlerhaft waren.

Grundsätzlich sollte dieser Rückgabewert nie vorkommen, da die Satensätze für die SEPA-Zahlungen bereits im Vorfeld geprüft wurden.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 94 SepaTools_XMLGetDataAZVN

## *94.1 Zweck der Funktion*

Unter der Voraussetzung, dass vorher eine XML-Datei erzeugt wurde kann der Inhalt der XML-Datei als Speicherabbild (Byte-Array) ausgelesen werden.

Der Ablauf stellt sich dabei folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CreateAZVN | Initialisierung der Verarbeitung |
| - Aufruf von SepaTools_WriteAZVN | Dieser Aufruf kann fast beliebig oft wieder holt werden (Begrenzung durch den verfügbaren Hauptspeicher des Rechners).<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Zahlungsverkehrsauftrages übergeben. |
| … | |
| - Aufruf von SepaTools_CloseAZVN | Die Anwendung wird beendet und die Datei wird auf den angegebenen Datenträger geschrieben. |
| - Aufruf von SepaTools_XMLGetDataAZVN | Der Inhalt der XML-Datei wird als Byte-Array im Speicher übergeben. |

## *94.2 Funktionsaufruf*

**int SepaTools_XMLGetDataAZVN(char *Data, int *Size, char *DataSepa, int *SizeSepa)**

## *94.3 Parameter*

***Data***

Nach dem Aufruf der Funktion zeigt der Pointer „Data“ auf den Speicherbereich, ab dem sich der Inhalt der AZV-XML-Datei befindet. Über die Funktion SepaTools_XMLGetDataAZVN wird die Adresse des Speicherbereichs in die Variable „Data“ kopiert.

Bitte beachten:

Der erforderliche Speicherbereich wird von der API belegt. Die Anwendung braucht/darf hier keinen Speicherbereich belegen.

Der Speicherbereich für die zu übergebenden Daten wird von der API verwaltet. Bei jedem Aufruf dieser Funktion wird ein zuvor belegter Speicherbereich erst mal freigegeben, bevor ein neuer Speicherbereich belegt wird.

Beim Beenden der API über SepaTools_Free wird ein u.U. noch belegter Speicherbereich endgültig freigegeben.

***Size***

In dieser Variablen wird die Größe der Daten zurückgegeben. Die Daten sind mit einer abschließenden Null (NULL) abgeschlossen. Der hier zurückgegebene Wert für die Größe des Speicherbereichs ist daher um ein Byte größer als die tatsächlichen Daten.

***DataSepa*** Nach dem Aufruf der Funktion zeigt der Pointer „Data“ auf den Speicherbereich, ab dem sich der Inhalt der SEPA-XML-Datei befindet. Über die Funktion SepaTools_XMLGetDataAZVN wird die Adresse des Speicherbereichs in die Variable „DataSepa“ kopiert.

Ansonsten siehe Variable Data.

***SizeSepa*** In dieser Variablen wird die Größe der Daten (SEPA-XML) zurückgegeben. Die Daten sind mit einer abschließenden Null (NULL) abgeschlossen. Der hier zurückgegebene Wert für die Größe des Speicherbereichs ist daher um ein Byte größer als die tatsächlichen Daten.

### *94.4 Returncodes*

Ist der Rückgabewert positiv, so ist dieser bitcodiert (OR-Verbindung) und hat dabei folgende Bedeutung:

1 Es sind keine AZV-Datensätze vorhanden. Der entsprechende Speicherbereich ist nicht belegt.

2 Es sind keine SEPA-Datensätze vorhanden. Der entsprechende Speicherbereich ist nicht belegt.

Ansonsten hat der Rückgabewert folgende Bedeutung:

0 Alles verlief erfolgreich

-1 Um den Inhalt der XML-Datei als Byte-Array auszugeben, ist es erforderlich dass bei SepaTools_CreateAZVN für den Namen der XML-Datei ein Leerstring übergeben wurde. Dies ist hier nicht geschehen.

Wenn Sie diese Funktion nutzen möchten, muss bei SepaTools_CreateAZVN als Dateiname für die XML-Datei ein Leerstring ausgegeben werden.

-2 Der Speicherbereich für die Übergabe der Daten wurde nicht belegt. Möglicherweise stimmt die Reihenfolge der Funktionsaufrufe nicht.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 95 Auslesen von Camt- und Pain.002-Nachrichten

Die bisherigen Kontoauszugsformate (MT940, MT942, DTI) werden sukzessive durch die Camt-Formate (XML) abgelöst. Zusätzlich gibt es das neue Format Pain.002 für Statusinformationen (z.B. Rücklastschriften oder Sammler). Folgende Formate wurden zu diesem Zweck definiert:

| | |
|---|---|
| Camt.052 | Saldenreport und untertägige Umsätze |
| Camt.053 | Tagesauszug |
| Camt.054 | Sammelbuchungsdatei - Sammlerauflösung |

| | |
|---|---|
| Pain.002 | Statusreport |

Diese drei unterschiedlichen Camt-Formate und das Pain.002-Format können über SepaTools ausgelesen werden und werden automatisch erkannt. Wird eine ungültige Datei übergeben, so bricht die Verarbeitung mit einem negativen Rückgabecode ab.

Die möglichen Informationen in einer Camt-Datei sind sehr umfangreich. Selten werden alle Informationen benötigt. Der Ansatz von SepaTools ist es daher, die am häufigsten benötigten Informationen zur Verfügung zu stellen. Nicht bei allen Camt-Formaten, werden alle (die gleichen) Informationen zur Verfügung gestellt (z.B. fehlt beim Camt.054 in der Regel der Saldo).

Zur weiteren Verarbeitung können diese Datenformate mit SepaTools ausgelesen und in CSV-Dateien umgewandelt werden. Für die Verarbeitung von Camt-Formaten ist eine zusätzliche Lizenz von SepaTools für die Camt-Formate erforderlich. Ohne diese zusätzliche Lizenz kann die Camt-Funktionalität 60 Tage lang zu Testzwecken verwendet werden. Bitte beachten Sie hierzu den Lizenzvertrag.

Von den Bankprogrammen werden die Camt-Formate in der Regel als ZIP-Datei mit mehreren Einzeldateien geliefert. Die ZIP-Datei muss entpackt werden, um die einzelnen Nachrichten dann einzulesen. Auch dies kann automatisiert mit SepaTools erfolgen.

Wenn mehrere Camt-Dateien in einer ZIP-Datei enthalten sind, so werden in die gleiche CSV-Datei die Umsätze unterschiedlicher Konten und u.U. mehrerer Tage geschrieben. Über die einzelnen Felder können die einzelnen Umsätze aber genau zugeordnet werden. Es die gleiche Vorgehensweise, wie bisher bei den MT940-Dateien.

Die Steuerung der einzelnen Felder erfolgt über ein Feld, in dem die gewünschten Positionen der einzelnen werte hinterlegt sind.

Im Downloadbereich unter www.sepa-tools.de stehen Beispieldateien zur Verfügung.

# 96 SepaTools_ConvertCamtToCSV

## *96.1 Zweck der Funktion*

Die Funktion liest die Camt-Datei (camt.052, camt.053 und camt.054) aus und konvertiert die Datensätze in eine CSV-Datei.

## *96.2 Funktionsaufruf*

**int SepaTools_ConvertCamtToCSV(const CamtCSVStruct *CamtCSV)**

## *96.3 Parameter*

***CamtCSV*** Über diesen Parameter werden über einen Pointer auf einen Speicherbereich alle erforderlichen Variablen an die Funktion übergeben.

Die Struktur ist im Folgenden dargestellt.

struct CamtCSVStruct

| | | | |
|---|---|---|---|
| { | char | InputFile[255+1] | Der Pfad und der Dateiname für die Camt-Datei |
| | char | OutputFile[255+1] | Der Pfad und der Dateiname für die zu erzeug. CSV-Datei |
| | int | ZIP | Wert=0 wenn XML-Datei, Wert=1 wenn ZIP-Datei [1)] |
| | int | Sammler | Wert=0 keine Sammlerauflösung, Wert=1 Sammler [2)] |
| | int | Beteiligter | Siehe Erläuterung [3)] |
| | char | Credit[30+1 | Bezeichnung für Haben-Buchungen [4)] |
| | char | Debit[30+1] | Bezeichnung für Soll-Buchungen [5)] |
| | char | Trenn[1+1] | Trennzeichen für die CSV-Datei [6)] |
| | char | Grenze[1+1] | Optionale Begrenzung der Felder in der für die CSV-Datei [7)] |
| | char | Tausend[1+1] | Tausenderseparator der Felder in der für die CSV-Datei [8)] |
| | char | Komma[1+1] | Kommadarstellung der Beträge in der für die CSV-Datei [9)] |
| | int | CSVFeld[150] | Positionsangaben für die Felder in der CSV-Datei [10)] |
| | char | LibPath[200+1] | Optionaler Pfad für die Datei DUNZIP32.dll [11)] |
| | int | XML-Art | 0=Camt-Format, 1=Pain.002-Format |
| | char | Reserve[1000] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Von den Bankprogrammen werden die Camt-Formate in der Regel als ZIP-Datei mit mehreren Einzeldateien geliefert. Die ZIP-Datei muss entpackt werden, um die einzelnen Nachrichten dann einzulesen. Auch dies kann automatisiert mit SepaTools erfolgen.

   Wenn mehrere Camt-Dateien in einer ZIP-Datei enthalten sind, so werden in die gleiche CSV-Datei die Umsätze unterschiedlicher Konten und u.U. mehrerer Tage geschrieben. Über die einzelnen Felder können die einzelnen Umsätze aber genau zugeordnet werden. Es die gleiche Vorgehensweise, wie bisher bei den MT940-Dateien.

   Belegen Sie bitte dieses Feld mit dem Wert 1, wenn es sich um eine ZIP-Datei handelt.

   Wird dieses Feld mit 0 belegt, so erfolgt trotzdem eine Prüfung (über die Signatur in der Datei), ob es sich um eine ZIP-Datei handelt. Ist dies der Fall, so wird der Wert dieses Feldes intern auf 1 gesetzt.

2) Die Formate Camt.052 und Camt.053 können je nach den Einstellungen des Kreditinstituts Einzelposten eines Sammlers beinhalten, oder auch nicht. Wenn der Umsatz Sammlerinformationen enthält und Sie dieses Feld mit dem Wert 1 belegen, so werden die einzelnen Positionen des Sammlers wie einzelne Umsätze aufgelistet.

   Die gesamte Betragssumme des Sammlers taucht dann in den Datensätzen nicht mehr auf, da ja alle Einzelposten des Sammlers aufgelistet werden.

   Ist dieses Feld mit dem Wert 0 belegt, so werden die einzelnen Posten des Sammlers nicht aufgelistet.

   **Hinweis:**

   **Bitte sprechen Sie mit Ihrer Bank, bzgl. der Sammlerauflösung im Camt.052 und Camt.053 Datenformat.**

3) Werden Sammlerpositionen innerhalb des Camt.053 Formats dargestellt, so werden sowohl der beteiligte Debitor und der beteiligte Kreditor ausgewiesen.

   Häufig ist es aber nicht erforderlich, dass beide Werte dargestellt werden. Handelt es sich um Gutschriften, so ist der am Zahlungsverkehr beteiligte zwangsweise der Debitor. Handelt es sich im Kontoauszug um Belastungen, so ist der im Zahlungsverkehr Beteiligte zwangsweise der Creditor.

   Um die Auswertung der Felder zu erleichtern, können Sie dieses Feld mit dem Wert 1 belegen. In diesem Fall werden in den Feldern für **Camt-Debt-Name**, **Camt-Debt-Id**, **Camt-Debt-IBAN** und **Camt-Debt-Abw** immer die Daten des Zahlungsbeteiligten, unabhängig davon, ob Debitor oder Kreditor ausgewiesen. In der Praxis ist das immer das Gegenstück zum Kontoinhaber.

4) Wenn Sie dieses Feld mit einer Zeichenkette belegen, so können Sie den standardmäßigen Begriff für Habenbuchungen (CRDT) mit einem individuellen Wert überschreiben (z.B. mit dem Pluszeichen oder dem Wert H für Habenbuchung, oder mit einem beliebigen Wert mit bis zu 30 Zeichen Länge).

5) Wenn Sie dieses Feld mit einer Zeichenkette belegen, so können Sie den standardmäßigen Begriff für Sollbuchungen (DBIT) mit einem individuellen Wert überschreiben (z.B. mit dem Minuszeichen oder dem Wert S für Sollbuchung, oder mit einem beliebigen Wert mit bis zu 30 Zeichen Länge).

6) Geben Sie hier bitte optional an, mit welchem Trennzeichen die einzelnen Felder innerhalb der zu erzeugenden CSV-Datei getrennt werden sollen. Für dieses Feld stehen folgende Möglichkeiten zur Verfügung:

   **Komma**

   Als Trennzeichen wird dann das einfache Komma verwendet.

   **Strichpunkt**

   Wenn Sie hier einen Strichpunkt angeben, so wird als Trennzeichen der Strichpunkt verwendet.

**Tabulator**

Um ein Tabulatorzeichen als Trennzeichen zu verwenden, geben Sie bitte bei diesem Schlüsselbegriff den Wert „Tab“ (ohne Anführungszeihen) an. Die Angabe kann in Groß- oder Kleinbuchstaben erfolgen. Aufgrund dieser Angabe wird intern das richtige Tabulatorzeichen eingesetzt.

Wenn Sie ein anderes Trennzeichen, oder kein Trennzeichen als die hier angegebenen verwenden, wird intern das Trennzeichen automatisch mit dem Strichpunkt belegt.

7) Bei der Ausgabe von durch Trennzeichen getrennten Datenfeldern können Sie für die Datenfelder auch ein Begrenzungszeichen festlegen. Dies wird häufig für Textfelder verwendet.

Es stehen Ihnen folgende Möglichkeiten zur Verfügung:

**Kein Trennzeichen**

Das ist der Standardwert. Wenn Sie hier nichts angeben, wird kein Trennzeichen verwendet.

**Hochkomma**

Sie können hier als Trennzeichen ein Hochkomma angeben.

**Anführungszeichen**

Wenn Sie ein Anführungszeichen als Trennzeichen verwenden wollen, so geben Sie es bitte hier an.

8) Häufig werden Beträge mit Tausender-Separatoren dargestellt. Auch hier wird zum Teil der Punkt (.) oder das Komma (,) verwendet. Mit der Belegung dieses Feldes können Sie angeben, welcher Tausender-Separator (Punkt oder Komma) verwendet werden soll.

Soll kein Tausender-Separator verwendet werden, so können sie den Buchstaben **K** übergeben.

Wird das Feld nicht belegt, so wird intern der Wert K eingesetzt (kein Tausender-Separator).

9) Über dieses Feld wird definiert, wie das Dezimalkomma bei den Beträgen dargestellt wird. Je nach dem für welche Weiterverarbeitung die Daten bestimmt sind, kann es sich um den Punkt (.) oder das Komma (,) handeln.

Wird dieses Feld nicht belegt, so ist standardmäßig das Komma vorgegeben.

**Achtung:**

**Wenn Sie bei den Feldern „Komma“ und „Tausend“ den gleichen Wert zuweisen, so wird intern der Wert für das Dezimalkomma auf Komma (,) und der Wert für den Tausender-Separator auf K (kein Separator) gesetzt.**

10) Über dieses 10stellige Feld wird definiert, an welcher Stelle die einzelnen Werte in der CSV-Datei stehen sollen.

Durch die individuelle Steuerung des Aufbaus der CSV-Datei ist es auch nicht erforderlich, dass die einzelnen Felder in einer direkten Reihenfolge belegt werden. Zulässig ist auch die Generierung von Leerfeldern, in dem Sie Sprünge in der Definition zulassen.

Durch das Auslassen einzelner Positionen werden Leerfelder eingefügt, die ggfls. vom Importprogramm benötigt werden.

Geben Sie immer an der definierten Feldposition die Position des Feldes an, das es bei der zu erzeugenden CSV-Datei einnehmen soll.

**Beispiel für die Belegung des Feldes CSVFeld:**

| | |
|---|---|
| Datum des Auszugs | Position 1 |
| Name des Kontoinhabers | Position 2 |
| Anfangssaldo | Position 5 |
| Buchungsbetrag | Position 7 |
| Buchung Soll/Haben | Position 8 |

int CSVFeld[150] [0,1,0,0,0,2,0,0,0,0,0,0,5,0,0,0,0,0,7,8, . . . der Rest ist 0]

Der erste Eintrag der CSV-Datei sieht damit folgendermaßen aus:

```
27.12.2013;Testkonto Nummer 1;;;33,06;;2,00;DBIT
```

**Hinweis:**

**Es können nur Camt-Formate oder das Pain.002-Format ausgelesen werden. Eine Vermischung dieser beiden Formate ist nicht möglich.**

CSVFeld[1] Geben Sie hier die Position des Feldes für die Angabe des Camt-Formats in der CSV-Datei an. Gültige Werte für das Camt-Format sind 52, 53 und 54.

CSVFeld[2] Geben Sie hier die Position des Feldes für das Datum des Kontoauszugs (Erstellung) innerhalb der CSV-Datei an.

CSVFeld[3] Geben Sie hier die Position des Feldes für die Erstellungszeit des Kontoauszugs innerhalb der CSV-Datei an.

CSVFeld[4] Geben Sie hier die Position des Feldes für die Nummer des Kontoauszugs (rechtlich relevante Auszugsnummer) innerhalb der CSV-Datei an. Diese Auszugsnummer wird häufig nicht angegeben.

CSVFeld[5] Geben Sie hier die Position des Feldes für die laufende Sequenz des Kontoauszugs innerhalb der CSV-Datei an. Es handelt sich hierbei um eine, innerhalb des Jahres, fortlaufende Nummer des Kontoauszugs. Diese muss angegeben sein.

CSVFeld[6] Geben Sie hier die Position des Feldes für den Namen des Kontoinhabers innerhalb der CSV-Datei an.

CSVFeld[7] Geben Sie hier die Position des Feldes für den BIC des Kontoinhabers innerhalb der CSV-Datei an.

CSVFeld[8] Geben Sie hier die Position des Feldes für die IBAN des Kontoinhabers innerhalb der CSV-Datei an.

CSVFeld[9] Geben Sie hier die Position des Feldes für den Namen der Bank des Kontoinhabers innerhalb der CSV-Datei an.

CSVFeld[10] Geben Sie hier die Position des Feldes für die Währung des Kontos innerhalb der CSV-Datei an.

CSVFeld[11] Geben Sie hier die Position des Feldes für das Datum des Anfangssaldos dieses Kontoauszugs innerhalb der CSV-Datei an. Im Normalfall sollte dieses Datum mit dem Erstellungsdatum des Kontoauszugs identisch sein.

CSVFeld[12] Geben Sie hier die Position des Feldes für das Datum des Endsaldos dieses Kontoauszugs innerhalb der CSV-Datei an. Im Normalfall sollte dieses Datum mit dem Erstellungsdatum des Kontoauszugs und mit dem Datum des Anfangssaldos identisch sein.

CSVFeld[13] Geben Sie hier die Position des Feldes für den Anfangssaldo dieses Kontoauszugs innerhalb der CSV-Datei an. Es handelt sich hierbei um den Saldo **vor** Buchung des Umsatzes. Auf Grund der in der Camt-Datei übergebenen Daten (Anfangssaldo) wird dieser Saldo für jede Einzelposition berechnet.

CSVFeld[14] Geben Sie hier die Position des Feldes für den Endsaldo dieses Kontoauszugs innerhalb der CSV-Datei an. Es handelt sich hierbei um den Saldo **nach** Buchung des Umsatzes. Auf Grund der in der Camt-Datei übergebenen Daten (Anfangssaldo) wird dieser Saldo für jede Einzelposition berechnet.

CSVFeld[15] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll/Haben des Anfangssaldos dieses Kontoauszugs innerhalb der CSV-Datei an. Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Belegung der Felder Credit und Debit modifiziert werden.

CSVFeld[16] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll/Haben des Endsaldos dieses Kontoauszugs innerhalb der CSV-Datei an. Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Felder Credit und Debit modifiziert werden.

**Bitte beachten:**

**Bei der Sammleraufstellung im Camt.054 sind keine Salden enthalten. Die in den oben aufgeführten 6 Schlüsselbegriffe sind daher ungültig (als Saldo wird immer 0,00 unterstellt) und sollten bei der Auswertung des Formats camt.054 nicht verwendet werden.**

CSVFeld[17] Geben Sie hier die Position des Feldes für das Buchungsdatum des Umsatzes innerhalb der CSV-Datei an.

CSVFeld[18] Geben Sie hier die Position des Feldes für die Valuta (Wertstellung) des Umsatzes innerhalb der CSV-Datei an.

CSVFeld[19] Geben Sie hier die Position des Feldes für den Betrag des Umsatzes innerhalb der CSV-Datei an.

CSVFeld[20] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll/Haben des Umsatzes in diesem Kontoauszugs innerhalb der CSV-Datei an. Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Schlüsselbegriffe Credit und Debit modifiziert werden.

CSVFeld[21] Geben Sie hier die Position des Feldes für den beteiligten Creditor (Name) zu diesem Umsatz in der CSV-Datei an.

CSVFeld[22] Geben Sie hier die Position des Feldes für den beteiligten abweichenden Creditor (Name) zu diesem Umsatz in der CSV-Datei an.

CSVFeld[23] Geben Sie hier die Position des Feldes für eine optionale ID des beteiligten Creditors an. Diese ID kann eine Kontonummer sein, wenn es sich um keine SEPA-Zahlung handelt.

CSVFeld[24] Geben Sie hier die Position des Feldes für eine optionale IBAN des beteiligten Creditors an. Die IBAN ist i.d.R. angegeben, wenn es sich um eine SEPA-Zahlung handelt.

CSVFeld[25] Geben Sie hier die Position des Feldes für den beteiligten Debitors (Name) zu diesem Umsatz in der CSV-Datei an.

CSVFeld[26] Geben Sie hier die Position des Feldes für den beteiligten abweichenden Debitor (Name) zu diesem Umsatz in der CSV-Datei an.

CSVFeld[27] Geben Sie hier die Position des Feldes für eine optionale ID des beteiligten Debitors an. Diese ID kann eine Kontonummer sein, wenn es sich um keine SEPA-Zahlung handelt.

CSVFeld[28] Geben Sie hier die Position des Feldes für eine optionale IBAN des beteiligten Debitors an. Die IBAN ist i.d.R. angegeben, wenn es sich um eine SEPA-Zahlung handelt.

CSVFeld[29] Geben Sie hier die Position des Feldes für die erste Zeile des Verwendungszwecks dieses Umsatzes an. Es werden max. 3 Zeilen Verwendungszweck mit einer Länge von jeweils 140 Zeichen unterstützt.

CSVFeld[30] Geben Sie hier die Position des Feldes für die zweite Zeile des Verwendungszwecks dieses Umsatzes an. Es werden max. 3 Zeilen Verwendungszweck mit einer Länge von jeweils 140 Zeichen unterstützt.

CSVFeld[31] Geben Sie hier die Position des Feldes für die dritte Zeile des Verwendungszwecks dieses Umsatzes an. Es werden max. 3 Zeilen Verwendungszweck mit einer Länge von jeweils 140 Zeichen unterstützt.

CSVFeld[32] Innerhalb eines Tagesauszugs (camt.053) oder der untertätigen Saldeninformation (camt.052) können unterschiedliche Salden mitgeliefert werden. Diese Salden werden über einen Code gekennzeichnet.

SepaTools unterstützt die Übergabe von bis zu fünf unterschiedlichen Salden in einer Camt-Nachricht. Dieses Feld steht für das Datum des **ersten** gefundenen Saldos. In der Praxis werden häufig zwei Salden (Anfangssaldo und Schlusssaldo) übergeben.

CSVFeld[33] Geben Sie hier die Position des Feldes für den **ersten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an. Der Saldo bezieht sich auf die Gesamtheit der Buchungen in diesem Auszug.

CSVFeld[34] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll und Haben für den **ersten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an.

Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Felder Credit und Debit modifiziert werden.

CSVFeld[35] Geben Sie hier die Position des Feldes für die Kennzeichnung des **ersten** Saldos an.

Folgende Werte sind zugelassen:

| | |
|---|---|
| CLBD | Endsaldo |
| CLAV | Valutensaldo zum angegebenen Datum |
| FWAV | Zukünftiger Valutensaldo zum angegebenen Datum |
| ITBD | Zwischensaldo im Buchungstag |
| PRCD | Anfangssaldo |

**Bitte beachten:**

**Bei der Sammleraufstellung im Camt.054 sind keine Salden enthalten. Die in den oben aufgeführten 6 Schlüsselbegriffe sind daher ungültig (als Saldo wird immer 0,00 unterstellt) und sollten bei der Auswertung des Formats camt.054 nicht verwendet werden.**

CSVFeld[36] Geben Sie hier die Position des Feldes für das Datum des **zweiten** Saldos dieses Kontoauszugs innerhalb der CSV-Datei an.

CSVFeld[37] Geben Sie hier die Position des Feldes für den **zweiten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an. Der Saldo bezieht sich auf die Gesamtheit der Buchungen in diesem Auszug.

CSVFeld[38] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll und Haben für den **zweiten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an.

Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Felder Credit und Debit modifiziert werden.

CSVFeld[39] Geben Sie hier die Position des Feldes für die Kennzeichnung des **zweiten** Saldos an.

CSVFeld[40] Geben Sie hier die Position des Feldes für das Datum des **dritten** Saldos dieses Kontoauszugs innerhalb der CSV-Datei an.

CSVFeld[41] Geben Sie hier die Position des Feldes für den **dritten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an. Der Saldo bezieht sich auf die Gesamtheit der Buchungen in diesem Auszug.

CSVFeld[42] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll und Haben für den **dritten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an.

Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Felder Credit und Debit modifiziert werden.

CSVFeld[43] Geben Sie hier die Position des Feldes für die Kennzeichnung des **dritten** Saldos an.

CSVFeld[44] Geben Sie hier die Position des Feldes für das Datum des **vierten** Saldos dieses Kontoauszugs innerhalb der CSV-Datei an.

CSVFeld[45] Geben Sie hier die Position des Feldes für den **vierten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an. Der Saldo bezieht sich auf die Gesamtheit der Buchungen in diesem Auszug.

CSVFeld[46] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll und Haben für den **vierten** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an.

Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Felder Credit und Debit modifiziert werden.

CSVFeld[47] Geben Sie hier die Position des Feldes für die Kennzeichnung des **vierten** Saldos an.

CSVFeld[48] Geben Sie hier die Position des Feldes für das Datum des **fünften** Saldos dieses Kontoauszugs innerhalb der CSV-Datei an.

CSVFeld[49] Geben Sie hier die Position des Feldes für den **fünften** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an. Der Saldo bezieht sich auf die Gesamtheit der Buchungen in diesem Auszug.

CSVFeld[50] Geben Sie hier die Position des Feldes für die Kennzeichnung von Soll und Haben für den **fünften** Saldo dieses Kontoauszugs innerhalb der CSV-Datei an.

Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo).

Diese Bezeichnungen können über die Felder Credit und Debit modifiziert werden.

CSVFeld[51] Geben Sie hier die Position des Feldes für die Kennzeichnung des **fünften** Saldos an.

CSVFeld[52] Geben Sie hier bitte die Position des Feldes für die laufende Nummer der Buchung in diesem Kontoauszug an. Die Buchungen werden laufend durchnummeriert.

Wenn in einer Buchung mehrere Posten vorhanden sind (Sammlerauflösung), so haben die einzelnen Posten in dem Sammler auf Grund dieses Feldes die gleiche laufende Nummer. Siehe nächstes Feld.

CSVFeld[53] Geben Sie hier bitte die Position des Feldes für die laufende Transaktion innerhalb eines Sammlers bei Sammlerauflösung. Die einzelnen Transaktionen innerhalb eines Sammlers werden laufend durchnummeriert.

Über dieses und das vorgehende Feld wird der Zusammenhang zwischen der Buchung auf dem Konto und den einzelnen Posten eines Sammlers hergestellt.

CSVFeld[54] Geben Sie hier die Position des Feldes für die EndToEnd-ID der Zahlung an. Die EndToEnd-ID ist in der Regel nur bei Sammlerauflösungen auf Transaktionsebene verfügbar.

CSVFeld[55] Geben Sie hier bitte die Position des Feldes für den Buchungscode, bzw. Textschlüssel der Zahlung an (Feld <BkTxCd><Prtry><Cd>). Das Ergebnis dieses Schlüsselbegriffes besteht aus mehreren teilen, die in den nachfolgenden Feldern aufgeschlüsselt werden.

Zur Erläuterung siehe bitte nächstes Kapitel.

CSVFeld[56] Geben Sie hier bitte die Position des Feldes für den Untercode „SWIFT-Transaction-Code“ der Zahlung an. Zur Erläuterung siehe bitte nächstes Kapitel.

CSVFeld[57] Geben Sie hier bitte die Position des Feldes für den Untercode „Geschäftsvorfallcode, GVC“ der Zahlung an. Zur Erläuterung siehe bitte nächstes Kapitel.

CSVFeld[58] Geben Sie hier bitte die Position des Feldes für den Untercode „Primanota-Nummer“ der Zahlung an. Zur Erläuterung siehe bitte nächstes Kapitel.

CSVFeld[59] Geben Sie hier bitte die Position des Feldes für den Untercode „DTA-Textschlüsselergänzung“ der Zahlung an. Zur Erläuterung siehe bitte nächstes Kapitel.

CSVFeld[60] Geben Sie hier die Position des Feldes für das Mandat an, das u.U. im Datensatz (bei Lastschriften) enthalten ist. Das Mandat kann sowohl bei der Buchung beim Zahlungsempfänger, als auch bei der Buchung des Zahlungspflichtigen vorhanden sein.

CSVFeld[61] Geben Sie bitte hier die Position des Feldes für den BIC bei zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist nur bei Lastschriftrückgaben oder Überweisungsrückgaben gefüllt.

CSVFeld[62] Geben Sie bitte hier die Position des Feldes für den Rückgabegrund (z.B. MD06 für Lastschriftrückgaben wegen Widerspruchs) bei zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist nur bei Lastschriftrückgaben oder Überweisungsrückgaben gefüllt.

CSVFeld[63] Geben Sie bitte hier die Position des Feldes für den Kontotyp (z.B. CASH) bei zurückgegebenen Zahlungen innerhalb der CSV-Datei an.

CSVFeld[64] Geben Sie bitte hier die Position des Feldes für die Message-ID bei zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld wird häufig nur in Verbindung mit Sammlern verwendet. Damit kann der Sammler referenziert werden.

CSVFeld[65] Geben Sie bitte hier die Position des Feldes für die Payment-ID bei zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld wird häufig nur in Verbindung mit Sammlern verwendet. Damit kann der Sammler referenziert werden.

CSVFeld[66] Geben Sie bitte hier die Position des Feldes für die Angabe der Posten im Sammler bei zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld wird nur in Verbindung mit Sammlern verwendet.

CSVFeld[67] Geben Sie bitte hier die Position des Feldes für eine Zusatzinformation zur zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist häufig nicht gefüllt.

CSVFeld[68] Geben Sie bitte hier die Position des Feldes für den BIC des beteiligten Creditors zur zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist häufig nicht gefüllt.

CSVFeld[69] Geben Sie bitte hier die Position des Feldes für den Namen der Bank des beteiligten Creditors zur zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist häufig nicht gefüllt.

CSVFeld[70] Geben Sie bitte hier die Position des Feldes für den BIC des beteiligten Debitors zur zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist häufig nicht gefüllt.

CSVFeld[71] Geben Sie bitte hier die Position des Feldes für den Namen der Bank des beteiligten Debitors zur zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist häufig nicht gefüllt.

CSVFeld[72] Geben Sie bitte hier die Position des Feldes für die Referenz des ersten (1.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[73] Geben Sie bitte hier die Position des Feldes für den Code des ersten (1.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[74] Geben Sie bitte hier die Position des Feldes für die Referenz des zweiten (2.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[75] Geben Sie bitte hier die Position des Feldes für den Code des zweiten (2.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[76] Geben Sie bitte hier die Position des Feldes für die Referenz des dritten (3.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[77] Geben Sie bitte hier die Position des Feldes für den Code des dritten (3.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[78] Geben Sie hier bitte die Position des Feldes des ursprünglichen Betrags an. Dies ist der ursprüngliche Betrag ohne Gebühren. Der tatsächliche Betrag der Transaktion mit Gebühren steht im Feld CSVFeld[19].

CSVFeld[79] Es kann sein, dass die berechneten Gebühren aufgeschlüsselt sind. Max. 3 Gebührenaufschlüsselungen werden ausgelesen.

Geben Sie bitte hier die Position des Feldes für die erste Gebühr an.

CSVFeld[80] Es kann sein, dass die berechneten Gebühren aufgeschlüsselt sind. Max. 3 Gebührenaufschlüsselungen werden ausgelesen.

Geben Sie bitte hier die Position des Feldes für die zweite Gebühr an.

CSVFeld[81] Es kann sein, dass die berechneten Gebühren aufgeschlüsselt sind. Max. 3 Gebührenaufschlüsselungen werden ausgelesen.

Geben Sie bitte hier die Position des Feldes für die dritte Gebühr an.

CSVFeld[82] Ab der DK-Version 3.0 werden auf Sammlerebene und auf Transaktionsebene zusätzliche Domaininformationen (SEPA-Buchungsschlüssel) geliefert.

Geben Sie über diesen Schlüsselbegriff bitte die Position für den Domain-Code auf Sammlerebene an.

CSVFeld[83] Ab der DK-Version 3.0 werden auf Sammlerebene und auf Transaktionsebene zusätzliche Domaininformationen (SEPA-Buchungsschlüssel) geliefert.

Geben Sie über diesen Schlüsselbegriff bitte die Position für den Family-Code auf Sammlerebene an.

CSVFeld[84] Ab der DK-Version 3.0 werden auf Sammlerebene und auf Transaktionsebene zusätzliche Domaininformationen (SEPA-Buchungsschlüssel) geliefert.

Geben Sie über diesen Schlüsselbegriff bitte die Position für den Sub-Family-Code auf Sammlerebene an.

CSVFeld[85] Ab der DK-Version 3.0 werden auf Sammlerebene und auf Transaktionsebene zusätzliche Domaininformationen (SEPA-Buchungsschlüssel) geliefert.

Geben Sie über diesen Schlüsselbegriff bitte die Position für den Bank-Transaktionscode auf Sammlerebene an.

CSVFeld[86] Ab der DK-Version 3.0 werden auf Sammlerebene und auf Transaktionsebene zusätzliche Domaininformationen (SEPA-Buchungsschlüssel) geliefert.

Geben Sie über diesen Schlüsselbegriff bitte die Position für den Domain-Code auf Transaktionsebene an.

CSVFeld[87] Ab der DK-Version 3.0 werden auf Sammlerebene und auf Transaktionsebene zusätzliche Domaininformationen (SEPA-Buchungsschlüssel) geliefert.

Geben Sie über diesen Schlüsselbegriff bitte die Position für den Family-Code auf Transaktionsebene an.

CSVFeld[88] Ab der DK-Version 3.0 werden auf Sammlerebene und auf Transaktionsebene zusätzliche Domaininformationen (SEPA-Buchungsschlüssel) geliefert.

Geben Sie über diesen Schlüsselbegriff bitte die Position für den Sub-Family-Code auf Transaktionsebene an.

CSVFeld[89] Geben Sie bitte hier die Position des Feldes für eine Zusatzinformation (Ntry-Ebene) zur zurückgegebenen Zahlungen innerhalb der CSV-Datei an. Dieses Feld ist häufig nicht gefüllt. Für das Feld stehen 200 Zeichen zur Verfügung.

CSVFeld[90] Geben Sie bitte hier die Position des Feldes für den Prtry-Eintrag des ersten (1.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[91] Geben Sie bitte hier die Position des Feldes für den Prtry-Eintrag des zweiten (2.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[92] Geben Sie bitte hier die Position des Feldes für den Prtry-Eintrag des dritten (3.) ausgelesenen Blocks des strukturierten Verwendungszwecks innerhalb der CSV-Datei an. Dieses Feld ist nur gefüllt, wenn anstelle des unstrukturierten Verwendungszwecks ein strukturierter Verwendungszweck ausgegeben wird.

CSVFeld[93] Geben Sie bitte hier die Position des Feldes für eine Zusatzinformation auf Dtls-Ebene innerhalb der CSV-Datei an. Dieses Feld ist häufig nicht gefüllt. Für das Feld stehen 200 Zeichen zur Verfügung.

CSVFeld[94] Geben Sie bitte hier die Position des Feldes für eine Bank-Service-Id an. Dieses Feld gibt es auf der Ebene von Ntry und auf der Ebene NtryDtls (als Referenz). Wenn beide Felder belegt sind, so wird die Referenz Vorrang.

CSVFeld[95] Geben Sie bitte hier die Position des Feldes für eine optionale Transaktionsreferenz (Zahlungsreferenz) an.

CSVFeld[96] Geben Sie bitte hier die Position des Feldes für ein eventuelles Stornokennzeichen an. Bei Storno wird der Wert 1 geliefert, ansonsten 0.

CSVFeld[97] Wenn bei einem Sammler die Einzelbuchungen ausgegeben werden, so wird hier die Betragssumme des Sammlers geliefert.

**Ab hier folgen die Belegungen für das Format Pain.002**

CSVFeld[111] Geben Sie bitte hier die Position des Feldes für „Art der Datei“ an. Hier wird immer der Wert 02 übergeben.

CSVFeld[112] Geben Sie bitte hier die Position des Feldes für Message-Id innerhalb der Pain.002-Datei an.

CSVFeld[113] Geben Sie bitte hier die Position des Feldes für das Erstellungsdatum der Datei an. Das Datum wird im Format TT.MM.JJJJ übergeben.

CSVFeld[114] Geben Sie bitte hier die Position des Feldes für die Uhrzeit der Erstellung der Datei an. Die Uhrzeit wird im Format HH.MM.SS übergeben.

CSVFeld[115] Geben Sie bitte hier die Position des Feldes für den BIC der Bank an, die die Datei erstellt hat an.

CSVFeld[116] Geben Sie bitte hier die Position des Feldes für die ursprüngliche Message-Id (Message-Id der ursprünglichen Zahlung) an.

CSVFeld[117] Geben Sie bitte hier die Position des Feldes für die Anzahl der in dieser Datei enthaltenen Datensätze an.

CSVFeld[118] Geben Sie bitte hier die Position des Feldes für die Gesamtsumme der in dieser Datei enthaltenen Beträge (Summe der Beträge pro Datensatz) Datensätze an.

CSVFeld[119] Geben Sie bitte hier die Position des Feldes für die ursprüngliche Payment-Information-Id (Pm-Id der ursprünglichen Zahlung) an.

CSVFeld[120] Geben Sie bitte hier die Position des Feldes für die Status-Id der Zahlung (Datensatz) an. Dieser Wert wird häufig nicht geliefert.

CSVFeld[121] Geben Sie bitte hier die Position des Feldes für die ursprüngliche EndToEnd-Id (EndToEnd der ursprünglichen Zahlung) an.

CSVFeld[122] Geben Sie bitte hier die Position des Feldes für die ursprüngliche Message-Bezeichnung ursprünglichen Zahlung an (z.B. pain.008).

CSVFeld[123] Geben Sie bitte hier die Position des Feldes für den Status der Transaktion an (z.B. RJCT bei Rückgaben).

CSVFeld[124]Geben Sie bitte hier die Position des Feldes für den Namen des Rückgebers der Zahlung an. Dieses Feld wird sehr häufig nicht geliefert.

CSVFeld[125]Geben Sie bitte hier die Position des Feldes für den BIC der rückgebenden Bank an.

CSVFeld[126] Geben Sie bitte hier die Position des Feldes für den Rückgabecode (Rückgabegrund) der Zahlung (z.B. AC01).

CSVFeld[127] Geben Sie bitte hier die Position des Feldes für den Betrag der Zahlung an.

CSVFeld[128] Geben Sie bitte hier die Position des Feldes für das Ausführungsdatum der Zahlung an. Bei Überweisung ist es das tatsächliche Ausführungsdatum, bei Lastschriften das Fälligkeitsdatum der Lastschrift.

Das Datum wird im Format TT.MM.JJJJ angegeben.

CSVFeld[129] Geben Sie bitte hier die Position des Feldes für die CI (Gläubiger Identifikation) der Zahlung an.

CSVFeld[130] Geben Sie bitte hier die Position des Feldes für das Zahlungsinstrument der Zahlung an (z.B. B2B, CORE, COR1 SDD).

CSVFeld[131]Geben Sie bitte hier die Position des Feldes für die Sequenz der Lastschrift an (z.B. FRST, RCUR).

CSVFeld[132] Geben Sie bitte hier die Position des Feldes für das Mandat der Zahlung an.

CSVFeld[133]Geben Sie bitte hier die Position des Feldes für das Datum des Mandats der Zahlung an.

Das Datum wird im Format TT.MM.JJJJ angegeben.

CSVFeld[134] Geben Sie bitte hier die Position des Feldes für den Verwendungszweck der Zahlung an.

CSVFeld[135]Geben Sie bitte hier die Position des Feldes für einen u.U. vorhandenen abweichenden Namen des Creditors an.

CSVFeld[136] Geben Sie bitte hier die Position des Feldes für den Namen des beteiligten Creditors an.

CSVFeld[137] Geben Sie bitte hier die Position des Feldes für die IBAN des beteiligten des Creditors an.

CSVFeld[138] Geben Sie bitte hier die Position des Feldes für den BIC des beteiligten des Creditors an.

CSVFeld[139] Geben Sie bitte hier die Position des Feldes für einen u.U. vorhandenen abweichenden Namen des Debitors an.

CSVFeld[140] Geben Sie bitte hier die Position des Feldes für den Namen des beteiligten Debitors an.

CSVFeld[141] Geben Sie bitte hier die Position des Feldes für die IBAN des beteiligten des Debitors an.

CSVFeld[142] Geben Sie bitte hier die Position des Feldes für den BIC des beteiligten des Debitors an.

11) Wenn Sie mit ZIP-Dateien arbeiten wollen (die wiederum die Camt-Dateien enthalten), so ist hierfür die Datei DUNZIP32.DLL erforderlich. Diese muss auf dem Computer gefunden werden. Entweder Sie liegt im aktuellen Verzeichnis, oder die Datei wird über einen Pfad gefunden.

    Wenn Sie diese DLL woanders ablegen wollen, so können Sie hier optional einen Pfad für diese DLL übergeben. Die Datei wird dann in diesem Pfad gesucht.

## *96.4 Vorgabe zur Belegung des Feldes <BkTxCd><Prtry><Cd>*

Der Code besteht aus folgenden Teilen, die zusammen als String, verbunden mit jeweils “+” eingestellt werden:

- Vierstelliger SWIFT-Transaction-Code
- Geschäftsvorfallcode (GVC)
- Optional: Primanota-Nr. (maximal 10-stellig)
- DTA-Textschlüsselergänzung, falls darstellbar

Beispiele:

`<Cd>NDDT+109+9002/405+901</Cd>` Beispiel für eine SEPA-Lastschriftrückgabe

`<Cd>NDDT+009+9002/405+052</Cd>` Beispiel für eine DTA-Lastschriftrückgabe

Die Textschlüsselergänzung kann fehlen (z.B. bei SEPA-Zahlungen)

`<Cd>NTRF+116+9002/405</Cd>` Beispiel für eine SEPA-Überweisung

Sollte ein Zwischenteil (z.B. Primanota) fehlen, dann werden zwei Pluszeichen gesetzt, um die Lücke innerhalb des Strings zu signalisieren

`<Cd>NDDT+109++901</Cd>` Beispiel für eine SEPA-Lastschriftrückgabe

`<Cd>NTRF+116</Cd>` Beispiel für eine SEPA-Überweisung

## *96.5 Returncodes*

0 Alles verlief erfolgreich

-1 Die angegebene XML-Datei, bzw. ZIP-Datei konnte nicht geöffnet werden, bzw. wurde nicht gefunden.

-2 Der Pfad für die Ausgabedatei (die zu erstellende XML-Datei) ist formal ungültig (kleiner als zwei Zeichen).

-3 Die Pfadangabe (mit Dateinamen) für die Ausgabedatei (XML-Datei) ist zu lang (länger als 200 Zeichen).

-4 Das Laufwerk im Pfad für die Ausgabedatei ist nicht bereit.

-5 Das Verzeichnis, das für die Ausgabedatei angegeben wurde existiert nicht.

-6 Der Datenträger, der für die Ausgabedatei angegeben wurde ist schreibgeschützt.

-7 Die gewünschte XML-Datei konnte nicht erzeugt werden.

-8 Die angegebene XML-Datei konnte nicht geöffnet werden.

-9 Die angegebene XML-Datei entspricht keinem gültigem Format.

-11 Innerhalb der entpackten XML-Dateien (aus der ZIP-Datei) konnte eine Datei nicht geöffnet werden.

-12 Innerhalb der ZIP-Datei mit Camt-Informationen war eine Datei enthalten, die nicht den Camt-Formaten entspricht. Analog Pain.002.

-21 Die angegebene ZIP-Datei wurde nicht gefunden.

-22 Die Struktur der ZIP-Datei ist korrupt (zerstört). Die Datei kann nicht verwendet werden.

-23 Die Ausgabedateien konnten nicht erzeugt werden. Bitte prüfen Sie den Pfad, den Sie in der Funktion SepaTools_Init für das Temp-Verzeichnis übergeben haben.

-24 Sonstiger Fehler beim Entpacken der ZIP-Datei.

-25 Sie haben für die Datei DUNZIP32.DLL einen Pfad übergeben. Die Datei ist aber in diesem Pfad nicht vorhanden.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

## *96.6 Verarbeitung von ZIP-Dateien*

Über das Feld **ZIP** innerhalb der Struktur wird gesteuert, ob von einer ZIP-Datei (Wert=1) oder einer XML-Datei (Wert=0) ausgegangen wird. Handelt es sich um eine XML-Datei, so wird diese Datei direkt ausgewertet. Bei einer ZIP-Datei sieht der interne, technische Ablauf folgendermaßen aus.

- Ausgehend von dem temporären Verzeichnis (gesetzt durch den Aufruf SepaTools_Init), wird in diesem temporären Verzeichnis ein Unterverzeichnis mit dem Namen **SepaTools** erzeugt, wenn dieses nicht schon vorhanden ist.

- Vorsorglich werden alle Daten mit der Dateinamensergänzung*.xml in diesem Verzeichnis gelöscht.

- Die im Feld **Input-File** übergebene ZIP-Datei wird in das Verzeichnis **SepaTools** entpackt.

- Alle jetzt in diesem Verzeichnis liegenden Dateien mit der Dateinamenergänzung *.xml werden der Reihe nach so verarbeitet, als würden sie jedes Mal als Input-File übergeben. Anschließend werden die XML-Dateien in diesem Verzeichnis gelöscht.

  Obwohl dies in der Bankpraxis derzeit nicht vorkommt, könnten in der ZIP-Datei auch unterschiedliche Camt-Formate (52, 53 und 54) enthalten sein. Die Umsetzung würde korrekt funktionieren. In den umgesetzten Daten steht die Information über das Camt-Format zur Verfügung.

- Um diesen Prozess abwickeln zu können, sind neben dem Programm SepaBatch.exe noch die Dateien SepaUnZip.exe und DUnZip32.dll erforderlich. Diese müssen auf dem Rechner gefunden werden.

  Entweder sie liegen im aktuellen Arbeitsverzeichnis, oder sie sind über einen Pfad (Umgebungs-Variable Path=) erreichbar. Diese beiden Dateien können auch in das Systemverzeichnis von Windows kopiert werden.

# 97 Camt-Dateien und Pain.002-Dateien auslesen

SepaTools stellt Ihnen Funktionen zur Verfügung, mit der Camt-Dateien (camt.052, camt.053 und camt.054) und Camt.002-Dateien ausgelesen werden können und in einer definierten Struktur zur Weiterverarbeitung zur Verfügung gestellt werden.

Der logische Ablauf stellt sich folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CamtOpen | Initialisierung der Anwendung und Einlesen der Camt-Informationen in eine temporäre Datei. |
| - Aufruf von SepaTools_CamtRead | Dieser Aufruf kann bis zum Auftreten eines negativen Rückgabecodes mehrfach ausgeführt werden.<br><br>Pro Aufruf werden in der übergebenen Struktur die Daten eines einzelnen Camt-Umsatzes übergeben. |
| … | |
| - Aufruf von SepaTools_CamtClose | Die Anwendung wird beendet und der Speicher wird wieder freigegeben. Die temporäre Datei wird gelöscht. |

# 98 SepaTools_CamtOpen

## *98.1 Zweck der Funktion*

Mit dieser Funktion wird die Funktion initialisiert. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

Die angegebene Camt-Datei (oder ZIP-Datei) wird ausgelesen und in eine temporäre Datei geschrieben.

## *98.2 Funktionsaufruf*

**int SepaTools_CamtOpen(CamtOpenStruct *CamtOpen)**

## *98.3 Parameter*

***CamtOpen*** Hier wird ein Pointer auf eine Variable mit der Struktur CamtOpenStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

struct CamtOpenStruct

| | | | |
|---|---|---|---|
| { | char | InputFiled[255+1] | Laufwerk und Pfad für die Eingangsdatei. |
| | int | ZIP | Wert=0 wenn XML-Datei, Wert=1 wenn ZIP-Datei [1)] |
| | int | Sammler | Wert=0 keine Sammlerauflösung, Wert=1 Sammler [2)] |
| | int | Beteiligter | Siehe Erläuterung [3)] |
| | char | Credit[30+1] | Bezeichnung für Haben-Buchungen [4)] |
| | char | Debit[30+1] | Bezeichnung für Soll-Buchungen [5)] |
| | char | LibPath[200+1] | Optionaler Pfad für die Datei DUNZIP32.dll [6)] |
| | int | XMLArt | 0=Camt-Dateien, 1=Pain.002-Datei |
| | char | ZIPName[64+1] | Wird hier nicht verwendet (nur für CamtWriteOpen). |
| | int | CompressZIP | Wird hier nicht verwendet (nur für CamtWriteOpen). |
| | char | Reserve[931] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Von den Bankprogrammen werden die Camt-Formate in der Regel als ZIP-Datei mit mehreren Einzeldateien geliefert. Die ZIP-Datei muss entpackt werden, um die einzelnen Nachrichten dann einzulesen. Auch dies kann automatisiert mit SepaTools erfolgen.

   Wenn mehrere Camt-Dateien in einer ZIP-Datei enthalten sind, so werden in Sruktur die Umsätze unterschiedlicher Konten und u.U. mehrerer Tage geschrieben. Über die einzelnen Felder können die einzelnen Umsätze aber genau zugeordnet werden. Es die gleiche Vorgehensweise, wie bisher bei den MT940-Dateien.

   Belegen Sie bitte dieses Feld mit dem Wert 1, wenn es sich um eine ZIP-Datei handelt.

Wird dieses Feld mit 0 belegt, so erfolgt trotzdem eine Prüfung (über die Signatur in der Datei), ob es sich um eine ZIP-Datei handelt. Ist dies der Fall, so wird der Wert dieses Feldes intern auf 1 gesetzt.

2) Die Formate Camt.052, Camt.053 und Camt.054 können je nach den Einstellungen des Kreditinstituts Einzelposten eines Sammlers beinhalten, oder auch nicht. Wenn der Umsatz Sammlerinformationen enthält und Sie dieses Feld mit dem Wert 1 belegen, so werden die einzelnen Positionen des Sammlers wie einzelne Umsätze aufgelistet.

   Die gesamte Betragssumme des Sammlers taucht dann in den Datensätzen nicht mehr auf, da ja alle Einzelposten des Sammlers aufgelistet werden.

   Ist dieses Feld mit dem Wert 0 belegt, so werden die einzelnen Posten des Sammlers nicht aufgelistet.

   **Hinweis:**

   **Bitte sprechen Sie mit Ihrer Bank, bzgl. der Sammlerauflösung im Camt.052 und Camt.053 Datenformat.**

3) Werden Sammlerpositionen innerhalb des Camt.053 Formats dargestellt, so werden sowohl der beteiligte Debitor und der beteiligte Kreditor ausgewiesen.

   Häufig ist es aber nicht erforderlich, dass beide Werte dargestellt werden. Handelt es sich um Gutschriften, so ist der am Zahlungsverkehr beteiligte zwangsweise der Debitor. Handelt es sich im Kontoauszug um Belastungen, so ist der im Zahlungsverkehr Beteiligte zwangsweise der Creditor.

   Um die Auswertung der Felder zu erleichtern, können Sie dieses Feld mit dem Wert 1 belegen. In diesem Fall werden in den Feldern für **Camt-Debt-Name**, **Camt-Debt-Id**, **Camt-Debt-IBAN** und **Camt-Debt-Abw** immer die Daten des Zahlungsbeteiligten, unabhängig davon, ob Debitor oder Kreditor ausgewiesen. In der Praxis ist das immer das Gegenstück zum Kontoinhaber.

4) Wenn Sie dieses Feld mit einer Zeichenkette belegen, so können Sie den standardmäßigen Begriff für Habenbuchungen (CRDT) mit einem individuellen Wert überschreiben (z.B. mit dem Pluszeichen oder dem Wert H für Habenbuchung, oder mit einem beliebigen Wert mit bis zu 30 Zeichen Länge).

5) Wenn Sie dieses Feld mit einer Zeichenkette belegen, so können Sie den standardmäßigen Begriff für Sollbuchungen (DBIT) mit einem individuellen Wert überschreiben (z.B. mit dem Minuszeichen oder dem Wert S für Sollbuchung, oder mit einem beliebigen Wert mit bis zu 30 Zeichen Länge).

6) Wenn Sie mit ZIP-Dateien arbeiten wollen (die wiederum die Camt-Dateien enthalten), so ist hierfür die Datei DUNZIP32.DLL erforderlich. Diese muss auf dem Computer gefunden werden. Entweder Sie liegt im aktuellen Verzeichnis, oder die Datei wird über einen Pfad gefunden.

   Wenn Sie diese DLL woanders ablegen wollen, so können Sie hier optional einen Pfad für diese DLL übergeben. Die Datei wird dann in diesem Pfad gesucht.

   Bitte beachten: In der 64-Bit Version wird dieser Wert nicht verwendet.

### *98.4 Returncodes*

0 Alles verlief erfolgreich

-1 Die angegebene XML-Datei, bzw. ZIP-Datei konnte nicht geöffnet werden, bzw. wurde nicht gefunden.

-2 Die temporäre Datei könnte nicht erzeugt werden.

-8 Die angegebene XML-Datei konnte nicht geöffnet werden.

-9 Die angegebene XML-Datei entspricht keinem gültigen Camt-Format.

-11 Innerhalb der entpackten XML-Dateien (aus der ZIP-Datei) konnte eine Datei nicht geöffnet werden.

-12 Innerhalb der ZIP-Datei mit Camt-Informationen war eine Datei enthalten, die nicht den Camt-Formaten entspricht. analog Pain.002.

-21 Die angegebene ZIP-Datei wurde nicht gefunden.

-22 Die Struktur der ZIP-Datei ist korrupt (zerstört). Die Datei kann nicht verwendet werden.

-23 Die Ausgabedateien konnten nicht erzeugt werden. Bitte prüfen Sie den Pfad, den Sie in der Funktion SepaTools_Init für das Temp-Verzeichnis übergeben haben.

-24 Sonstiger Fehler beim Entpacken der ZIP-Datei.

-25 Sie haben für die Datei DUNZIP32.DLL einen Pfad übergeben. Die Datei ist aber in diesem Pfad nicht vorhanden.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 99 SepaTools_CamtRead

## *99.1 Zweck der Funktion*

Diese Funktion wird mehrfach aufgerufen. Bei jedem Aufruf wird die übergebene Struktur mit den Werten eines Umsatzes aus der Camt-Datei gefüllt. Ein zweiter Teil der Struktur wird bei Bedarf mit den Informationen der Pain.002-Datei gefüllt.

## *99.2 Funktionsaufruf*

**int SepaTools_CamtRead(int *First, CamtReadStruct *CamtRead)**

## *99.3 Parameter*

***First*** Übergeben Sie bitte beim ersten Aufruf der Funktion hier eine Variable mit dem Wert 1. Dies signalisiert der Funktion, dass die temporäre Datei geöffnet werden muss. Der Wert der Variablen ***First*** wird nach dem ersten Aufruf der Funktion intern automatisch auf den Wert 0 gesetzt.

***CamtRead*** Hier wird ein Pointer auf eine Variable mit der Struktur CamtReadStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

In der Struktur werden jeweils die Informationen eines Umsatzes aus der Camt-Datei abgelegt.

struct CamtReadStruct

| | | | |
|---|---|---|---|
| { | char | Art[2+1] | Typ der erkannten Camt-Datei (52, 53 oder 54) |
| | char | AuszugDatum[8+1] | Erstellungsdatum des Kontoauszugs im Format TTMMJJJJ |
| | char | AuszugZeit[6+1] | Erstellungszeit des Kontoauszugs im Format HHMMSS |
| | char | AuszugNummer[20+1] | Juristische Nummer des Auszugs |
| | char | Sequenz[20+1] | Laufende Sequenz des Auszugs |
| | char | KontoName[70+1] | Name des Kontoinhabers |
| | char | KontoBIC[11+1] | BIC des Kontos |
| | char | KontoIBAN[35+1 | IBAN des Kontos |
| | char | KontoBankName[70+1] | Name der Bank des Kontoinhabers |
| | char | KontoWhg[3+1] | Währung des Kontos (EUR) |
| | char | SaldoDat_A[8+1] | Datum des Anfangssaldos pro Ums. im Format TTMMJJJJ [1)] |
| | char | SaldoDat_E[8+1] | Datum des Endsaldos pro Umsatz im Format TTMMJJJJ [1)] |
| | char | SaldoA[12+1] | Anfangssaldo für diesen Umsatz in Cent [2)] |
| | char | SaldoE[12+1] | Endsaldo für diesen Umsatz in Cent [3)] |
| | char | SaldoSH_A[30+1] | Soll-/Haben-Kennzeichnung des Anfangssaldos [4)] |
| | char | SaldoSH_E[30+1] | Soll-/Haben-Kennzeichnung des Endsaldos [4)] |
| | char | BuDatum[8+1] | Buchungsdatum des Umsatzes im Format TTMMJJJJ |
| | char | Valuta[8+1] | Valuta des Umsatzes im Format TTMMJJJJ |
| | char | Betrag[12+1] | Betrag des Umsatzes (der Buchung) in Cent |
| | char | SH[30+1] | Soll-/Haben-Kennzeichnung des Umsatzes [5)] |
| | char | Cred_Name[70+1] | Name des an der Zahlung beteiligten Creditors |
| | char | Cred_Abw[70+1] | Abweich. Name des an der Zahlung beteiligten Creditors |
| | char | Cred_Id[35+1] | Id (z.B. Kto.-Nr.) des an der Zahlung beteiligten Creditors |
| | char | Cred_IBAN[35+1] | IBAN des an der Zahlung beteiligten Creditors |

| | | |
|---|---|---|
| char | Debt_Name[70+1] | Name des an der Zahlung beteiligten Debitors |
| char | Debt_Abw[70+1] | Abweich. Name des an der Zahlung beteiligten Debitors |
| char | Debt_Id[35+1] | Id (z.B. Kto.-Nr.) des an der Zahlung beteiligten Debitors |
| char | Debt_IBAN[35+1] | IBAN des an der Zahlung beteiligten Debitors |
| char | Zweck1[140+1] | Erste Zeile des Verwendungszwecks |
| char | Zweck2[140+1] | Zweite Zeile des Verwendungszwecks |
| char | Zweck3[140+1] | Dritte Zeile des Verwendungszwecks |
| char | T_SaldoDat_1[8+1] | Datum des 1. Tagessaldos im Auszug[6)] |
| char | T_Saldo_1[12+1] | Tagessaldo 1 des Auszugs in Cent [7)] |
| char | T_SaldoSH_1[30+1] | Soll-/Haben-Kennzeichnung des Saldos [8)] |
| char | T_SaldoCode_1[4+1] | Codierung/Bedeutung des Saldos [9)] |
| char | T_SaldoDat_2[8+1] | Datum des 2. Tagessaldos im Auszug[6)] |
| char | T_Saldo_2[12+1] | Tagessaldo 2 des Auszugs in Cent [7)] |
| char | T_SaldoSH_2[30+1] | Soll-/Haben-Kennzeichnung des Saldos [8)] |
| char | T_SaldoCode_2[4+1] | Codierung/Bedeutung des Saldos [9)] |
| char | T_SaldoDat_3[8+1] | Datum des 3. Tagessaldos im Auszug[6)] |
| char | T_Saldo_3[12+1] | Tagessaldo 3 des Auszugs in Cent [7)] |
| char | T_SaldoSH_3[30+1] | Soll-/Haben-Kennzeichnung des Saldos [8)] |
| char | T_SaldoCode_3[4+1] | Codierung/Bedeutung des Saldos [9)] |
| char | T_SaldoDat_4[8+1] | Datum des 4. Tagessaldos im Auszug[6)] |
| char | T_Saldo_4[12+1] | Tagessaldo 4 des Auszugs in Cent [7)] |
| char | T_SaldoSH_4[30+1] | Soll-/Haben-Kennzeichnung des Saldos [8)] |
| char | T_SaldoCode_4[4+1] | Codierung/Bedeutung des Saldos [9)] |
| char | T_SaldoDat_5[8+1] | Datum des 5. Tagessaldos im Auszug[6)] |
| char | T_Saldo_5[12+1] | Tagessaldo 5 des Auszugs in Cent [7)] |
| char | T_SaldoSH_5[30+1] | Soll-/Haben-Kennzeichnung des Saldos [8)] |
| char | T_SaldoCode_5[4+1] | Codierung/Bedeutung des Saldos [9)] |
| int | Num_Buchung | Laufende Nummer dieses Umsatzes [10)] |
| int | Num_Tx | Laufende Nr. der Transaktion in einer Sammlerauflösung [11)] |
| char | EndToEnd[35+1] | EndToEnd-Referenz, wenn angegeben [12)] |
| char | Code[35+1] | Buchungscode und/oder Textschlüssel [13)] |
| char | Code_Swift[4+1] | Vierstelliger SWIFT-Transaction-Code |
| char | Code_GVC[3+1] | Geschäftsvorfallcode (GVC) auf Transaktionsebene. |
| char | Code_PN[10+1] | Optionale Primanota-Nummer (max. 10stellig) |
| char | Code_DTA[3+1] | DTA-Textschlüsselergänzung, falls darstellbar |
| char | Mandat[35+1] | Mandat bei Lastschriften [14)] |
| char | R_BIC[11+1] | BIC bei Rücklastschriften/Rückweisungen [15)] |
| char | R_Grund[4+1] | Codierter Grund der Rückgabe [16)] |
| char | KontoTyp[4+1] | Type des Kontos, z.B. CASH |
| char | MessageID[35+1] | Message-ID, wenn angegeben (i.d.R. bei Sammlern) |
| char | PaymentID[35+1] | Payment-ID, wenn angegeben (i.d.R. bei Sammlern) |
| int | SammlerTx | Anzahl Sammlerposten |
| char | ZusatzInfo[35+1] | Zusatzinformation zu einer Transaktion |
| char | Cred_BIC[11+1] | Der BIC des beteiligten Creditors, soweit angegeben |
| char | Cred_BankName[35+1] | Der Name der Bank des beteiligten Creditors, soweit ang. |
| char | Debt_BIC[11+1] | Der BIC des beteiligten Debitors, soweit angegeben |
| char | Debt_BankName[35+1] | Der Name der Bank des beteiligten Debitors, soweit ang. |
| char | MsgNameId[35+1] | Ein optionaler Referenzname auf eine andere Camt-Datei |
| char | MsgId[35+1] | Eine opt. Referenz auf den Inhalt einer anderen Camt-Datei |
| char | StrdCode1[4+1] | Der Code bei der Ausgabe eines strukt. Verw.-Zwecks [17)] |
| char | StrdRef1[35+1] | Die Referenz bei der Ausgabe eines strukt. Verw.-Zwecks [17)] |
| char | StrdCode2[4+1] | Der Code bei der Ausgabe eines strukt. Verw.-Zwecks [17)] |
| char | StrdRef2[35+1] | Die Referenz bei der Ausgabe eines strukt. Verw.-Zwecks [17)] |
| char | StrdCode3[4+1] | Der Code bei der Ausgabe eines strukt. Verw.-Zwecks [17)] |

| | | |
|---|---|---|
| char | StrdRef3[35+1] | Die Referenz bei der Ausgabe eines strukt. Verw.-Zwecks [17)] |
| char | Cred_CI[35+1] | Die CI des beteiligten Creditors, soweit angegeben. |
| char | Debt_CI[35+1] | Die CI des beteiligten Debitors, soweit angegeben. |
| char | UrSprBetrag[12+1] | Der urspr. Betrag einer Lastschrift bei Rückgabe in Cent. |
| char | Gebuehr1[12+1] | Gebühr für Lastschriftrückgabe in Cent. |
| char | Gebuehr2[12+1] | Gebühr für Lastschriftrückgabe in Cent. |
| char | Gebuehr3[12+1] | Gebühr für Lastschriftrückgabe in Cent. |
| char | EntryDomCode[4+1] | BkTx-Domaincode auf Sammlerebene |
| char | EntryFamCode[4+1] | BkTx-Familycode auf Sammlerebene |
| char | EntrySubFamCode[4+1] | BkTx-Sub-Familycode auf Sammlerebene |
| char | EntryBkTxCode[35+1] | BkTx-Code auf Sammlerebene |
| char | TxDomCode[4+1] | BkTx-Domaincode auf Transaktionsebene [13] |
| char | TxFamCode[4+1] | BkTx-Familycode auf Transaktionsebene [13] |
| char | TxSubFamCode[4+1] | BkTx-Sub-Familycode auf Transaktionsebene [13] |
| char | ZusatzInfoLang[200+1] | Zusatzinformation auf Ntry-Ebene, bis zu 200 Zeichen |
| char | StrdPrtry1[35+1] | Der Prtry-Eintrag im strukturierten Verw.-Zweck [17)] |
| char | StrdPrtry2[35+1] | Der Prtry-Eintrag im strukturierten Verw.-Zweck [17)] |
| char | StrdPrtry3[35+1] | Der Prtry-Eintrag im strukturierten Verw.-Zweck [17)] |
| char | ZusatzInfoTx[200+1] | Zusatzinformation auf Dtls-Ebene, bis zu 200 Zeichen |
| char | AcctServiceRef[35+1] | Optional die Bankreferenz [19)] |
| char | TransactionId[35+1] | Optionale Transaktions-Id [20)] |
| int | RvslInd | Wert=1 wenn Stornobuchung |
| char | Whg[3+1] | Die Währung des Umsatzes |
| char | BatchBetrag[12+1] | Der Betrag eines u.U. vorhandenen Sammlers |
| char | NachrichtMsgId[35+1] | Die Message Id der gesamten Nachricht (Datei) |
| char | SammlerSumme[12+1] | Summe des Sammlers bei Sammelbuchungen [21)] |
| char | PaymentInfoId[35+1] | Optionale PaymentInfoId |
| char | Reserve1[122] | Reserve zur späteren Verwendung. |

**Ab hier die Felder für die Pain.002-Datei**

| | | |
|---|---|---|
| char | PainMsgId[35+1] | Die Message-Id der Pain.002-Datei |
| char | PainDatum[8+1] | Erstellungsdatum im Format TTMMJJJJ |
| char | PainZeit[6+1] | Erstellungszeit im Format HHMMSS |
| char | PainBICA[11+1] | BIC der erstellenden Bank |
| char | PainOrgMsgId[35+1] | Message-Id der ursprünglichen Nachricht |
| int | PainNbTx | Anzahl der Buchungen in der Pain.002-Datei |
| char | PainSummeBetrag[12+1] | Gesamtsumme der Buchungen in der Pain.002-Datei in Cent |
| char | PainOrgPmInfo[35+1] | Payment-Info-Id der ursprünglichen Nachricht |
| char | PainStatusId[35+1] | Status-Id der Pain-Nachricht (wird häufig nicht geliefert) |
| char | PainOrgEndToEnd[35+1] | EndToEnd-Id der ursprünglichen Nachricht |
| char | PainOrgMessage[20+1] | Bezeichnung der ursprünglichen Nachricht, z.B. pain.008 |
| char | PainTxStatus[4+1] | Status der Transaktion (z.B. RJCT bei Rückgaben) |
| char | PainRName[35+1] | Name des Rückgebers (wird häufig nicht angegeben) |
| char | PainRBIC[11+1] | BIC der zurückgebenden Bank |
| char | PainRCode[4+1] | Code der Rückgabe, z.B. AC01 |
| char | PainBetrag[12+1] | Betrag der Transaktion in Cent |
| char | PainAusDatum[8+1] | Ausführungsdatum der Transaktion, Format TTMMJJJJ [18)] |
| char | PainCI[35+1] | CI innerhalb der Pain.002-Nachricht |
| char | PainInstrument[4+1] | Zahlungsinstrument (z.B. B2B, CORE, COR1 SDD) |
| char | PainSequenz[4+1] | Sequenz der Lastschrift (z.B. FRST, RCUR usw.) |
| char | PainMandat[35+1] | Mandat der Lastschrift |
| char | PainMandatDat[8+1] | Datum des Mandats im Format TTMMJJJJ |
| char | PainZweck[140+1] | Verwendungszweck |

| | | |
|---|---|---|
| char | PainCredUlt[35+1] | Abweichender Creditor, soweit vorhanden |
| char | PainCredName[35+1] | Name des Creditors |
| char | PainCredIBAN[35+1] | IBAN des Creditors |
| char | PainCredBIC[11+1] | BIC des Creditors |
| char | PainDebtUlt[35+1] | Abweichender Debitor, soweit vorhanden |
| char | PainDebtName[35+1] | Name des Debitors |
| char | PainDebtIBAN[35+1] | IBAN des Debitors |
| char | PainDebtBIC[11+1] | BIC des Debitors |
| char | PainAddtInfo[70+1] | Zusätzliche Informationen in der Pain Struktur |
| char | AddtlInfo1[35+1] | Zusätzliche Inform. zum strukturierten Verwendungszweck |
| char | AddtlInfo2[35+1] | Zusätzliche Inform. zum strukturierten Verwendungszweck |
| char | AddtlInfo3[35+1] | Zusätzliche Inform. zum strukturierten Verwendungszweck |
| char | DateiName[80+1] | Dateiname der auszuwertenden Datei [22)] |
| char | Reserve2[171] | Reserve zur späteren Verwendung |

}

Zusätzliche Erklärungen:

1) Im Normalfall sollte dieses Datum mit dem Erstellungsdatum des Kontoauszugs identisch sein.

2) Es handelt sich hierbei um den Saldo **vor** Buchung des Umsatzes. Auf Grund der in der Camt-Datei übergebenen Daten (Anfangssaldo) wird dieser Saldo für jede Einzelposition berechnet.

3) Es handelt sich hierbei um den Saldo **nach** Buchung des Umsatzes. Auf Grund der in der Camt-Datei übergebenen Daten (Anfangssaldo) wird dieser Saldo für jede Einzelposition berechnet.

4) Standardmäßig ist dieses Feld mit CRDT (Haben-Saldo) bzw. DBIT (Soll-Saldo) belegt. Diese Bezeichnungen können über die Belegung der Felder Credit und Debit modifiziert werden.

   **Bitte beachten:**

   **Bei der Sammleraufstellung im Camt.054 sind keine Salden enthalten. Die in den oben aufgeführten 6 Schlüsselbegriffe sind daher ungültig (als Saldo wird immer 0,00 unterstellt) und sollten bei der Auswertung des Formats camt.054 nicht verwendet werden.**

5) Standardmäßig ist dieses Feld mit CRDT (Haben-Saldo) bzw. DBIT (Soll-Saldo) belegt. Diese Bezeichnungen können über die Belegung der Felder Credit und Debit modifiziert werden.

6) Innerhalb eines Tagesauszugs (camt.053) oder der untertätigen Saldeninformation (camt.052) können unterschiedliche Salden mitgeliefert werden. Diese Salden werden über einen Code gekennzeichnet.

   SepaTools unterstützt die Übergabe von bis zu fünf unterschiedlichen Salden in einer Camt-Nachricht. In der Praxis werden häufig zwei Salden (Anfangssaldo und Schlusssaldo) übergeben.

7) Der Saldo (1 bis 5) bezieht sich auf die Gesamtheit der Buchungen in diesem Auszug.

8) Standardmäßig sind dies die Bezeichnungen CRDT (Haben-Saldo) und DBIT (Soll-Saldo). Diese Bezeichnungen können über die Felder Credit und Debit modifiziert werden.

9) Der Typ des Saldos wird über eine Kennung dargestellt.
Folgende Werte sind zugelassen:

| | |
|---|---|
| CLBD | Endsaldo |
| CLAV | Valutensaldo zum angegebenen Datum |
| FWAV | Zukünftiger Valutensaldo zum angegebenen Datum |
| ITBD | Zwischensaldo im Buchungstag |
| PRCD | Anfangssaldo |

10) Die Buchungen werden laufend durchnummeriert. Wenn in einer Buchung mehrere Posten vorhanden sind (Sammlerauflösung), so haben die einzelnen Posten in dem Sammler auf Grund dieses Feldes die gleiche laufende Nummer. Siehe nächstes Feld.

11) Die einzelnen Transaktionen innerhalb eines Sammlers werden laufend durchnummeriert. Über dieses und das vorgehende Feld wird der Zusammenhang zwischen der Buchung auf dem Konto und den einzelnen Posten eines Sammlers hergestellt.

12) Die EndToEnd-ID ist in der Regel nur bei Sammlerauflösungen auf Transaktionsebene verfügbar.

13) Der Buchungscode ist häufig eine Kombination aus einem Swift-Code und dem bisher bekannten Textschlüssel (z.B. NMSC+201). Innerhalb der nächsten 4 Felder wird dieser aus mehreren Positionen zusammengesetzte Buchungscode aufgeschlüsselt.

    Ab der DK-Version 3.0 werden auch Domaininformationen (SEPA Buchungsschlüssel) sowohl auf Sammlerebene (Ntry) und auf Transaktionsebene (TxDtls) geliefert.

14) Das Mandat kann sowohl bei der Buchung beim Zahlungsempfänger, als auch bei der Buchung des Zahlungspflichtigen vorhanden sein.

15) Dieses Feld ist nur bei Lastschriftrückgaben oder Überweisungsrückgaben gefüllt. Es handelt sich hier um ein optionales Feld.

16) In diesem Feld wird der Rückgabegrund (z.B. MD06 für Lastschriftrückgaben wegen Widerspruchs) der Zahlung angegeben. Dieses Feld ist nur bei Lastschriftrückgaben oder Überweisungsrückgaben gefüllt.

17) Anstelle des nicht strukturierten Verwendungszwecks kann in der Camt-Datei auch ein Strukturierter Verwendungszweck mit den Werten Code, Referenz und Prtry ausgegeben werden. Dieser Block kann grundsätzlich mehrfach vorkommen. SepaTools unterstützt das bis zu dreifache Vorkommen dieses Blocks.

18) Bei Überweisung ist es dies das tatsächliche Ausführungsdatum, bei Lastschriften das Fälligkeitsdatum der Lastschrift.

19) Hier wird die optionale Bank-Service-Id ausgegeben. Dieses Feld gibt es auf der Ebene von Ntry und auf der Ebene NtryDtls (als Referenz). Wenn beide Felder belegt sind, so wird die Referenz Vorrang.

20) Hier kann optional eine Trasaktionsrefernz (Zahlungsreferenz) ausgegeben werden.

21) Wenn bei Sammelbuchungen die Einzelbuchungen mit ausgegeben werden sollen (Sammler=1 in der CamtOpen Struktur), dann wird hier die Summe des Sammlers angegeben.

22) Hier wird der Dateiname der gerade auszuwertenden Datei (bei ZIP-Dateien) übergeben. Wurde eine einzelne XML-Datei eingelesen, so entspricht dieser Wert dem Dateinamen, der in dr Funktion SepaTools_CamtOpen übergeben wurde.

    Wird allerdings eine gepackte Datei, mit mehreren XML-Dateien übergeben, so wird über dieses Feld der Dateiname der gerade ausgewerteten Datei übergeben (in der gepackten Datei, befinden sich in der Regel mehrere XML-Dateien).

### *99.4 Returncodes*

0 Alles verlief erfolgreich

-1 Die Funktion wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_CamtOpen auf.

-2 Die temporäre Datei könnte nicht geöffnet werden.

-3 Es ist ein Fehler beim Lesen der temporären Datei aufgetreten.

-4 Sie sind am Ende der Daten angekommen. Es stehen keine weiteren Datensätze mehr zur Verfügung.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 100 SepaTools_CamtClose

## *100.1 Zweck der Funktion*

Mit dieser Funktion wird die temporäre Datei wieder gelöscht und die belegten Speicherbereiche werden wieder freigegeben.

## *100.2 Funktionsaufruf*

**int SepaTools_CamtClose()**

## *100.3 Parameter*

keine.

## *100.4 Returncodes*

0 Alles verlief erfolgreich

-1 Die Funktion wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_CamtOpen auf.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 101 Verarbeitung von ZIP-Dateien

Über das Feld **ZIP** innerhalb der Struktur wird gesteuert, ob von einer ZIP-Datei (Wert=1) oder einer XML-Datei (Wert=0) ausgegangen wird. Handelt es sich um eine XML-Datei, so wird diese Datei direkt ausgewertet. Bei einer ZIP-Datei sieht der interne, technische Ablauf folgendermaßen aus.

- Ausgehend von dem temporären Verzeichnis (gesetzt durch den Aufruf SepaTools_Init), wird in diesem temporären Verzeichnis ein Unterverzeichnis mit dem Namen **SepaTools** erzeugt, wenn dieses nicht schon vorhanden ist.

- Vorsorglich werden alle Daten mit der Dateinamensergänzung*.xml in diesem Verzeichnis gelöscht.

- Die im Feld **Input-File** übergebene ZIP-Datei wird in das Verzeichnis **SepaTools** entpackt.

- Alle jetzt in diesem Verzeichnis liegenden Dateien mit der Dateinamenergänzung *.xml werden der Reihe nach so verarbeitet, als würden sie jedes Mal als Input-File übergeben. Anschließend werden die XML-Dateien in diesem Verzeichnis gelöscht.

  Obwohl dies in der Bankpraxis derzeit nicht vorkommt, könnten in der ZIP-Datei auch unterschiedliche Camt-Formate (52, 53 und 54) enthalten sein. Die Umsetzung würde korrekt funktionieren. In den umgesetzten Daten steht die Information über das Camt-Format zur Verfügung.

- Um diesen Prozess abwickeln zu können, ist neben dem Programm SepaTools DLL noch die Datei **DUnZip32.dll** erforderlich. Diese müssen auf dem Rechner gefunden werden.

  Entweder sie liegen im aktuellen Arbeitsverzeichnis, oder sie sind über einen Pfad (Umgebungs-Variable Path=) erreichbar. Diese beiden Dateien können auch in das Systemverzeichnis von Windows kopiert werden.

  Alternativ kann über die entsprechenden Funktionen ein Pfad für diese Datei übergeben werden.

  In der 64-Bit Version werden diese DLLs nicht benötigt.

# 102 Camt.053 und Camt.054 Dateien schreiben

Als Gegenstück zum Auslesen der Camt-Dateien, können nun auch Camt-Dateien geschrieben werden. Als Strukturen werden die gleichen Strukturen wie beim Lesen der Camt-Dateien verwendet.

Bitte beachten Sie, dass in den Strukturen viele Felder optonal sind. Einige werden innerhalb dieser Funktion von der API auch ignoriert.

Der logische Ablauf stellt sich folgendermaßen dar:

| | |
|---|---|
| - Aufruf von SepaTools_CamtWriteOpen | Initialisierung der Anwendung . |
| - Aufruf von SepaTools_CamtWrite | Dieser Aufruf kann bis zum Auftreten eines negativen Rückgabecodes mehrfach ausgeführt werden.<br><br>Pro Aufruf werden die in der angegebenen Struktur abgelegten Daten in eine emporäre Datei geschrieben. |
| … | |
| - Aufruf von SepaTools_CamtWriteClose | Die Anwendung wird beendet und der Speicher wird wieder freigegeben. Aus der temporären Datei wir eine Camt-Datei erstellt.<br><br>Anschließend wird die temporäre Datei wird gelöscht. |

Hinweis:

Mit der Demo-Version können maximal 5 Datensätze verarbeitet werden.

## *102.1 SepaTools_CamtWriteOpen*

## *102.2 Zweck der Funktion*

Mit dieser Funktion wird die Funktion initialisiert. Die entsprechenden Werte werden in der Struktur als Parameter übergeben.

## *102.3 Funktionsaufruf*

**int SepaTools_CamtWriteOpen(CamtOpenStruct *CamtOpen)**

## *102.4 Parameter*

| | |
|---|---|
| ***CamtOpen*** | Hier wird ein Pointer auf eine Variable mit der Struktur CamtOpenStruct übergeben. Die Struktur ist nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette. |

Bereits in dieser Funktion wird geprüft, ob in den angegebenen Pfad für die Exportdatei die auszugebende Datei auch geschrieben werden kann.

Es handelt sich hier um die gleiche Struktur, die auch für das Einleen von Camt-Informationen verwendet wird. Einige Felder werden für diese Funktion aber nicht belegt.

struct CamtOpenStruct

| | | | |
|---|---|---|---|
| { | char | Input_OutputFile[255+1] | In diesem Fall der Exportpfad für die zu erstellende Datei [4]. |
| | int | ZIP | Der Wert wird nicht verwendet. |
| | int | Sammler | Der Wert wird nicht verwendet |
| | int | Beteiligter | Der Wert wird nicht verwendet |
| | char | Credit[30+1] | Der Wert wird nicht verwendet |
| | char | Debit[30+1] | Der Wert wird nicht verwendet |
| | char | LibPath[200+1] | Optionaler Pfad für die Datei DZIP32.dll [1] |
| | int | XMLArt | 0=Camt.053-Dateien, 1=Pain.002-Datei, 2=Camt.054 [2] |
| | char | ZIPName[64+1] | Optionaler Name der ZIP-Datei [3] |
| | int | CompressZIP | 0=keine komprimierung, 1=Komprimierung der ZIP-Datei. |
| | char | Reserve[931] | Reserve zur späteren Verwendung |
| } | | | |

Zusätzliche Erklärungen:

1) Wenn Sie mit ZIP-Dateien arbeiten wollen (die wiederum die Camt-Dateien enthalten), so ist hierfür die Datei DZIP32.DLL erforderlich. Diese muss auf dem Computer gefunden werden. Entweder Sie liegt im aktuellen Verzeichnis, oder die Datei wird über einen Pfad gefunden.

   Wenn Sie diese DLL woanders ablegen wollen, so können Sie hier optional einen Pfad für diese DLL übergeben. Die Datei wird dann in diesem Pfad gesucht.

   Bitte beachten: In der 64-Bit Version wird dieser Wert nicht beachtet, da hier diese DLL nicht erforderlich ist.

2) Mit der Übergabe eines Wertes von 0 bis 2 wird hier die Art der zu erzeugenden Camt-Daei angegeben. Bitte beachten Sie, dass Pain.002 derzeit nicht unterstützt wird un die Übergabe des Wertes 1, die gleiche Bedeutung hat wie der Wert 0.

3) Eine ZIP-Datei kann unkomprimiert oder komprimiert erzeugt werden. Über diesen Parameter wird dies gesteuert.

4) Beim Lesen einer Camt-Datei ist hier der Dateiname der zu lesenden Datei anzugeben. Beim Schreiben wird hier der Pfad für die zu exportierenden Dateien angeben. Die Dateinamen werden intern automatisch nach dem Regelwerk der Deutschen Kreditwirtschaft ermittelt.

### *102.5 Returncodes*

0 Alles verlief erfolgreich

-2 Der Pfad für die hier zu erstellende Camt-Datei isst zu kurz (<2 Zeichen).

-3 Der Pfad für die zu erstellende Camt-Datei ist zu lang (>200 Zeichen).

-4 Das laufwerk für die Camt-Datei ist nicht bereit.

-5 Das Verzeichnis für die Camt-Datei (InputFile) existiert nicht.

-6 Das Verzeichnis für die Camt-Datei ist schreibgeschützt.

-7 Für die DLL DZIP32.DLL wurde ein optionaler Pfad angegeben. In diesem Pfad wurde die DLL aber nicht gefunden. In der 64-Bit Version ist diese DLL nicht erforderlich, daher kann dieser Fehlercode hier nicht auftreten.

-8 Die temporäre Datei konnte nicht erzeugt werden.angegebene XML-Datei konnte nicht geöffnet werden.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 103 SepaTools_CamtWrite

## *103.1 Zweck der Funktion*

Diese Funktion wird mehrfach aufgerufen. Bei jedem Aufruf wird der Inhalt der übergebenen Struktur mit den Werten eines Umsatzes in eine temporäre Datei geschrieben.

**Dabei wird die gleiche Strukur, wie beim Auslesen von Camt-Dateien verwendet.**

## *103.2 Funktionsaufruf*

**int SepaTools_CamtWrite( CamtReadStruct *CamtWrite)**

## *103.3 Parameter*

***CamtWrite*** Hier wird ein Pointer auf eine Variable mit der Struktur CamtReadStruct übergeben. Die Struktur ist die gleiche, wie beim Auslesen von Camt-Dateien.

Die Abweichungen sind nachfolgend dargestellt. Alle Strings werden nullterminiert übergeben. Die Länge der Zeichenarrays ist daher immer um eine Stelle länger als die tatsächliche Zeichenkette.

Die Struktur ist die gleiche, wie beim Auslesen von Camt-Dateien. Bitte verwenden Sie diese Struktur. Es werden ur die Felder dargestellt, die von dieser Funtion **nicht** verwendet werden.

| | | |
|---|---|---|
| char | Art[2+1] | Typ der erkannten Camt-Datei (52, 53 oder 54) |
| int | Num_Buchung | Laufende Nummer dieses Umsatzes [1)] |
| int | Num_Tx | Laufende Nr. der Transaktion in einer Sammlerauflösung [2)] |
| int | SammlerTx | Anzahl Sammlerposten |
| char | SammlerSumme[12+1] | Summe des Sammlers bei Sammelbuchungen |

Ebenso werden alle Felder, die für das Format Pain.002 in der Struktur angegeben sind, nicht verwendet.

Zusätzliche Erklärungen:

1) Die Buchungen werden laufend durchnummeriert. Wenn in einer Buchung mehrere Posten vorhanden sind (Sammlerauflösung), so haben die einzelnen Posten in dem Sammler auf Grund dieses Feldes die gleiche laufende Nummer. Siehe nächstes Feld.

2) Die einzelnen Transaktionen innerhalb eines Sammlers werden laufend durchnummeriert. Über dieses und das vorgehende Feld wird der Zusammenhang zwischen der Buchung auf dem Konto und den einzelnen Posten eines Sammlers hergestellt.

### *103.4 Returncodes*

0 Alles verlief erfolgreich

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 104 SepaTools_CamtWriteClose

## *104.1 Zweck der Funktion*

Mit dieser Funktion wird die Ausgabedatei erzeugt und die temporäre Datei wieder gelöscht, ebenso werden die belegten Speicherbereiche werden wieder freigegeben.

## *104.2 Funktionsaufruf*

**int SepaTools_CamtWriteClose()**

## *104.3 Parameter*

keine.

## *104.4 Returncodes*

0 Alles verlief erfolgreich

-300 bis -100

Hierbei handelt es sich um Probleme bei der Erstellung der ZIP-Datei. Bitte wenden Sie sich an den Hersteller.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

## *104.5 Praktische Hinweise*

Unter Verwendung dieser Funktion und der Funktion zum Einlesen von Camt-Dateien, können aus Camt.053-Dateien Camt.054-Dateien und umgekehrt erzeugt werden. Bitte beachten Sie aber, dass eine Verschachtelung der beiden Funktonen nicht möglich ist.

Folgender Ablauf wäre denkbar:

- Funktion CamtOpen
- Funktion CamtRead

…

- Schreiben der Struktur in eine temporäre Datei
- Funtion CamtClose

- Funktion CamtWriteOpen

…

- Lesen der temporären Datei
- Funktion CamtWrite
- Funktion CamtWriteClose

Da bei der Konvertierung von Camt.054 in Camt.053 keine Salden vorhanden sind, müsste zumindest einmal der Anfangssaldo über die Struktur CamtReadStruct angereichert werden.

# 105 DTA/DTI-Dateien in Camt.054-Dateien umsetzen

## *105.1 Zweck der Funktion*

Mit dieser Funktion können DTA/DTI-Dateien in Camt.054-Dateien konvertiert werden. Dies ist hilfreich, wenn die Banken zur Sammlerauflösung noch DTA/DTI-Dateien liefern, die Kundensoftware aber bereits Camt.054-Dateien erwartet.

Bei dieser Funktion werden keine Multi-DTA-Dateien unterstützt. Der Ablauf ist sehr einfach gehalten und geht von einer korrekten DTA/DTI-Datei aus.

Bitte beachten Sie in diesem Zusammenhang auch die Funktion *SepaTools_SetDTIMapping*.

## *105.2 Funktionsaufruf*

**int SepaTools_ConvertDTI(const char *InPath, const char *OutPath)**

## *105.3 Parameter*

| | |
|---|---|
| ***InPath*** | Hier wird ein Pointer auf den Pfad mit dem Dateienamen der Eingangsdatei (DTA/DTI-Datei) angegeben. |
| ***OutPath*** | Hier wird ein Pointer auf den Pfad mit dem Dateienamen der Ergebnisdatei (Camt.054, XML-Datei) angegeben. |

## *105.4 Returncodes*

| | |
|---|---|
| 0 | Alles verlief erfolgreich |
| -1 | Die angegebene DTA/DTI-Datei konnte nicht geöffnet werden. |
| -2 | Die angegebene DTADTI-Datei wurde nicht gefunden. Bitte überprüfen Sie Pfad und Dateinamen. |
| -3 | Bei der geöffneten Datei handelt es sich um keine DTA/DTI-Datei. |
| -4 | Der angegebene temporäre Pfad existiert nicht (siehe SepaTools_Init). |
| -5 | Die temporäre DTA/DTI-Datei konnte nicht erzeugt werden. |
| -6 | Fehler beim Schreiben der temporären Datei. |
| -7 | Die temporäre Datei konnte nicht geöffnet werden. |
| -8 | Fehler beim Lesen der temporären Datei. |
| -9 | Die DTA/DTI-Datei konnte nicht erfolgreich geschlossen werden. |
| -11 | Nicht behebbarer Fehler im ASatz (siehe DTAFehlerStruct). |
| -12 | Nicht behebbarer Fehler im CSatz (siehe DTAFehlerStruct). |
| -13 | Nicht behebbarer Fehler im ESatz (siehe DTAFehlerStruct). |
| -14 | ESatz gefunden, obwohl kein vorhergehender ASatz gefunden wurde. |
| -15 | ESatz gefunden, obwohl kein vorhergehender CSatz gefunden wurde. |
| -91 | Bei der Erzeugung der XML-Datei ist ein unerwartetes Problem aufgetreten. |
| -92 | Bei der Erzeugung der Kopfdaten in der XML-Datei ist ein unerwartetes Problem aufgetreten. |
| -93 | Beim Schreiben der XML-Datei ist ein unerwartetes Problem aufgetreten. |
| -94 | Beim Schließen der XML-Datei ist ein unerwartetes Problem aufgetreten. |

| | |
|---|---|
| -99 | Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz. |
| -999 | Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf. |

# 106 SepaTools_SetDTIMapping

## *106.1 Zweck der Funktion*

Im Rahmen der Konvertierung werden aus Bankleitzahl und Kontonummer entsprechend den IBAN-Regeln BIC und IBAN ermittelt. Dabei kann es grundsätzlich vorkommen, dass aufgrund der gegebenen Bankverbindung aufgrund der IBAN-Regeln kein BIC und keine IBAN ermittelt werden dürfen.

Um in diesem Fall den Prozess trotzdem durchlaufen lassen zu können wird eine Ersatz-BIC (z.B. Differenzkonto) verwendet. Wird kein Standardwert für den BIC angegeben, so wird intern der BIC BANKDEFFXXX verwendet.

Das Gleiche gilt für die IBAN. Wird kein Wert übergeben, so wird intern die IBAN DE56111111111234567890 verwendet.

Innerhalb einer Camt.054-Datei wird die fachliche Kennzeichnung einer Transaktion über einen Geschäftsvorfallcode, abgelegt in dem Feld <BkTxCd><Prtry><Cd> dargestellt. Über diesen Code erfolgt in den Folgeprogrammen die Weiterverarbeitung.

Diese Information ist aber in einer DTA/DTI-Datei nicht vorhanden. Anstelle dessen gibt es dort einen 5stelligen Textschlüssel, der in der „alten“ DTA-Welt eine ähnliche Bedeutung hat.

Um hier eine Überführung zu erhalten, kann mit dieser Funktion eine Mapping-Tabelle erstellt werden. Dafür werden der Textschlüssel und der dafür gewünscht Geschäftsvorfallcode angegeben.

Der Code besteht aus folgenden Teilen, die zusammen als String, verbunden mit jeweils “+” eingestellt werden:

- Vierstelliger SWIFT-Transaction-Code
- Geschäftsvorfallcode (GVC)
- Optional: Primanota-Nr. (maximal 10-stellig)
- DTA-Textschlüsselergänzung, falls darstellbar

Beispiele:

`<Cd>NDDT+109+9002/405+901</Cd>` SEPA-Lastschriftrückgabe

Sollte ein Zwischenteil (Primanota) fehlen, dann werden zwei Pluszeichen gesetzt, um die Lücke innerhalb des Strings zu signalisieren

`<Cd>NDDT+109++901</Cd>` SEPA-Lastschriftrückgabe

## *106.2 Funktionsaufruf*

```
int SepaTools_SetDTIMapping   (const char *TS,
                               const char *GVC,
                               const char *DefBIC,
                               const char *DefIBAN,
                               const char *DefGVC)
```

## *106.3 Parameter*

***TS*** Hier wird ein Pointer auf den DTA/DTI-Textschlüssel übergeben, der durch den GVC ersetzt werden soll.

***GVC*** Hier wird ein Pointer auf den gewünschten Geschäftsvorfallcode übergeben. Durch den mehrfachen Aufruf dieser Funktion, kann eine Tabelle aus Textschlüsseln und Geschäftsvorfallcodes aufgebaut werden.

Wird der gleiche Textschlüssel mehrfach angegeben, so wird ein vorher angegebener Textschlüssel überschrieben.

***DefBIC*** Im Rahmen der Konvertierung werden aus Bankleitzahl und Kontonummer entsprechend den IBAN-Regeln BIC und IBAN ermittelt. Dabei kann es grundsätzlich vorkommen, dass aufgrund der gegebenen Bankverbindung aufgrund der IBAN-Regeln kein BIC und keine IBAN ermittelt werden dürfen.

Um in diesem Fall den Prozess trotzdem durchlaufen lassen zu können wird eine Ersatz-BIC (z.B. Differenzkonto) verwendet. Dieser BIC ist über diesen Parameter anzugeben.

***DefIBAN*** Hier übergeben Sie bitte einen Pointer auf die IBAN für das Differenzkonto. Ansonsten gilt das oben gesagte.

***DefGVC*** Grundsätzlich sollte die Mapping-Tabelle mit den oben angegebenen Parametern gepflegt werden. Wird trotzdem kein passender Eintrag in der Mapping-Tabelle gefunden, so wird der Wert dieses Parameters verwendet.

Wird dieser Parameter nicht belegt, so wird intern der Wert NDDT verwendet.

Obwohl die letzten drei Parameter eigentlich nur einmal angeliefert werden müssen, wurden Sie der Einfachheit halber in der gleichen Funktion untergebracht, die für die Mappingtabelle der Textschlüssel zuständig ist.

Da diese Funktion im Normalfall mehrfach aufgerufen wird, würden bei einer unterschiedlichen Übergabe der letzten drei Parameter immer die drei zuletzt übergebenen Parameter gültig sein.

**Hinweis:**

**Diese Funktion muss vor der Funktion SepaTools_ConvertDTI aufgerufen werden!**

### *106.4 Returncodes*

0 Alles verlief erfolgreich

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 107 SepaTools_ConvertMT940

## *107.1 Zweck der Funktion*

Über diese Funktion kann eine MT940-Datei in eine oder mehrere Camt.053-Dateien konvertiert werden. Häufig sind in einer MT940-Datei Einträge für mehrere Konten und/oder mehrere Zeiträume enthalten.

Im Rahmen der Konvertierung werden entsprechend der Spezifikation eine oder mehrere Camt.053-Dateien erzeugt. Optional kann die Ausgabe der Ergebnisdateien in eine ZIP-Datei erfolgen.

Mit der Demoversion können max. 5 Datensätze konvertiert werden.

Im Rahmen der Konvertierung der MT940-Dateien in Camt.053-Dateien werden u.U. vorhandene Bankleitzahlen und Kontonummern entsprechend der dokumentierten IBAN-Regeln in BIC und IBAN umgesetzt.

Sollte dies nicht gelingen, so wird als BIC der Wert „XXXXDEXXXXX“ und für die IBAN der Wert „DE85999999999999999999“ eingesetzt.

## *107.2 Funktionsaufruf*

**int SepaTools_ConvertMT940 (const char *Source,**
**const char *DestPath,**
**const char *ZIPFile,**
**const char *LibPath,**
**const Int Compress)**

## *107.3 Parameter*

***Source*** Hier wird ein Pointer auf die zu konvertierende MT940-Datei übergeben.

***DestPath*** Bitte übergeben Sie hier einen Pointer auf das Ausgabeverzeichnis für die Ergebnisdateien. Das Verzeichnis muss existieren.

***ZIPFile*** Wenn Sie die Ausgabe der Ergebnisdateien in eine ZIP-Datei wünschen, so übergeben Sie hier bitte einen Pointer auf den Dateinamen der ZIP-Datei.

Wird hier kein Dateiname übergeben, so erfolgt die Ausgabe der Ergebnisdateien als Einzeldateien. Die Dateinamen der Ergebnisdateien werden entsprechend der Spezifikation automatisch gebildet.

***LibPath*** Für die Komprimierung der Einzeldateien in eine ZIP-Datei muss die DLL **DZIP32.DLL** gefunden werden. Diese kann sich in einem Windows-Systemverzeichnis befinden oder über die Umgebungsvariable Path= gefunden werden.

Optional können Sie hier einen abweichenden Pfad angeben, in dem diese DLL gesucht wird. Ansonsten übergeben Sie hier bitte einen Leerstring oder NULL.

***Compress*** Dieser Parameter wirkt nur im Zusammenhang mit ZIP-Dateien. Wird hier eine 0 übergeben, so werden die einzelnen Camt-Dateien zwar auch in einer ZIP-Datei gespeichert, aber nicht komprimiert.

Mit Übergabe einer 1 in diesem Parameter erfolgt eine Standard-Komprimierung in der ZIP-Datei. Die ZIP-Datei wird dann deutlich kleiner. Allerdings kann es sein, dass Folgeprogramme die Datei nicht entpacken kann.

### *107.4 Returncodes*

0 Alles verlief erfolgreich
-1 Das temporäre Verzeichnis existiert nicht.
-2 Die Eingangsdatei wurde nicht gefunden.
-3 Die temporäre Umsatzdatei konnte nicht erzeugt werden.
-4 Die temporäre Saldodatei konnte nicht erzeugt werden.
-5 Der Verzeichnisname für das Zielverzeichnis ist kleiner als 3 Zeichen.
-6 Der Verzeichnisname für das Zielverzeichnis ist größer als 200 Zeichen.
-7 Das Laufwerk für das Zielverzeichnis ist nicht bereit.
-8 Das Zielverzeichnis ist ungültig (kein gültiges Verzeichnis).
-9 Der Dateiname für die ZIP-Datei ist kleiner als 3 Zeichen.
-10 Die Eingangsdatei ist keine gültige MT940-Datei.
-14 Die Datei DZIP.dll wurde nicht gefunden.
-21 Die temporäre Umsatzdatei konnte nicht geöffnet werden.
-22 Die temporäre Saldodatei konnte nicht geöffnet werden.
-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.
-300 bis -100 Probleme mit der ZIP-Datei. Bitte informieren Sie den Hersteller.
-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 108 SepaTools_ConvertCamt54ToDTI

## *108.1 Zweck der Funktion*

Die Funktion liest die Camt.054-Datei aus und konvertiert die Datensätze in eine DTI-Datei. Die ist für die Verarbeitung von Sammlern hilfreich.

## *108.2 Funktionsaufruf*

**int SepaTools_ConvertCamt54ToDTI(const CamtDTIStruct *CamtDTI)**

## *108.3 Parameter*

***CamtDTI*** Über diesen Parameter werden über einen Pointer auf einen Speicherbereich alle erforderlichen Variablen an die Funktion übergeben.

Die Struktur ist im Folgenden dargestellt.

struct CamtDTIStruct

| | | | |
|---|---|---|---|
| { | char | InputFile[255+1] | Der Pfad und der Dateiname für die Camt-, oder ZIP-Datei |
| | char | OutputFile[255+1] | Der Pfad und der Dateiname für die zu erzeug. DTI-Datei [1)] |
| | char | LibPath[200+1] | Optionaler Pfad für die Datei DUNZIP32.dll [2)] |
| | int | ZIP | Wert=0 wenn XML-Datei, Wert=1 wenn ZIP-Datei [3)] |
| | char | GVCPfad[255+1] | Optionaler Pfad für die Mappingtabelle [4)] |
| | int | BIC | Anzahl Zeilen, die für den BIC verwendet werden sollen [5)] |
| | int | EREF | Anzahl Zeilen für die EndToEnd Referenz |
| | int | MREF | Anzahl Zeilen für die Mandatsreferenz |
| | int | CRED | Anzahl Zeilen für die CI (Creditor Identifier) |
| | int | OAMT | Anzahl Zeilen für den Ursprungsbetrag der Lastschrift |
| | int | COAM | Anzahl Zeilen für die Summe der Gebühren |
| | int | INFO | Anzahl Zeilen für zusätzliche Rückgabe Informationen |
| | int | SVWZ | Anzahl Zeilen für den Verwendungszweck |
| | int | ABWA | Anzahl Zeilen für abweichenden Auftraggeber |
| | int | ABWE | Anzahl Zeilen für abweichenden Empfänger |
| | int | Auto | Automatische Dateinamensbildung [6)] |
| | char | FailBLZ[8+1] | Ersatz-Bankleitzahl [7)] |
| | char | FailKonto[10+1] | Ersatz-Kontonummer [8)] |
| | int | Soll_GVC | Alle Soll-GVCs ignorieren [9)] |
| | int | Habenl_GVC | Alle Haben-GVCs ignorieren [9)] |
| | char | RefSearch[20+1] | Suchbegriff für Referenz [10)] |
| | int | RecSearchInclude | Suchbegriff im Zweck vor die Referenz stellen [10)] |
| | int | RefSearchLen | Max.Länge der gesuchten Referenz [10)] |
| | int | RefSearchOnlyTx | Referenz nur in “AddtlTxInf” suchen [10)] |
| | int | RefSearchAllways | Immer die gesamte Information verwenden [10)] |
| | int | StructFull | Siehe separaten Verweis [11)] |
| | int | GebuehrDetail | Siehe separaten Verweis [12)] |
| | int | ZweckStart | Siehe separaten Verweis [13] |
| | int | CoBaSonder | Sonderlösung Commerzbank [14)] |
| | char | Reserve[435] | Reserve zur späteren Verwendung. |
| } | | | |

Zusätzliche Erklärungen:

1) Der hier vergebene Dateiname wird intern automatisch mit einer laufenden Nummer ergänzt. Dies ist erforderlich, da sich in einer von der Bank gelieferten Datei (ZIP) mehrere unterschiedliche Camt-Dateien enthalten sein können. Für jede dieser Dateien wird eine eigene Ausgabedatei erzeugt.

2) Wenn Sie mit ZIP-Dateien arbeiten wollen (die wiederum die Camt-Dateien enthalten), so ist hierfür die Datei DUNZIP32.DLL erforderlich. Diese muss auf dem Computer gefunden werden. Entweder Sie liegt im aktuellen Verzeichnis, oder die Datei wird über einen Pfad gefunden.

   Wenn Sie diese DLL woanders ablegen wollen, so können Sie hier optional einen Pfad für diese DLL übergeben. Die Datei wird dann in diesem Pfad gesucht.

3) Von den Bankprogrammen werden die Camt-Formate in der Regel als ZIP-Datei mit mehreren Einzeldateien geliefert. Die ZIP-Datei muss entpackt werden, um die einzelnen Nachrichten dann einzulesen. Auch dies kann automatisiert mit SepaTools erfolgen.

   Wenn mehrere Camt-Dateien in einer ZIP-Datei enthalten sind, so werden in die gleiche CSV-Datei die Umsätze unterschiedlicher Konten und u.U. mehrerer Tage geschrieben. Über die einzelnen Felder können die einzelnen Umsätze aber genau zugeordnet werden. Es die gleiche Vorgehensweise, wie bisher bei den MT940-Dateien.

   Belegen Sie bitte dieses Feld mit dem Wert 1, wenn es sich um eine ZIP-Datei handelt.

   Wird dieses Feld mit 0 belegt, so erfolgt trotzdem eine Prüfung (über die Signatur in der Datei), ob es sich um eine ZIP-Datei handelt. Ist dies der Fall, so wird der Wert dieses Feldes intern auf 1 gesetzt.

4) Geben Sie hier bitte in diesem Feld den Pfad und den Dateinamen für eine individuelle Mappingtabelle zur Umsetzung von GVCs in Textschlüssel an.

   Es handelt sich hierbei um eine CSV-Datei, die unter diesem Pfad gefunden werden muss.

   Wird diese Datei gefunden, so erfolgt ein individuelles Mapping von GVC nach Textschlüssel.

   Die Mappingtabelle (CSV-Datei) hat beispielhaft folgenden Aufbau:

   ```
   GVG;TS;Reverse
   109;09000
   166;51000
   105;05000
   181;09001;1
   ```

   Wird bei Rücklastschriften ein Rückgabegrund erkannt, so wird dieser über eine interne Mappingtabelle als Textschlüsselergänzung zum Textschlüssel 09 zugeordnet. Diese Zuordnung hat aufgrund der klar definierten Textschlüsselergänzungen für Rückgabecodes Vorrang.

   In manchen Fällen werden Debitor und Creditor (z.B. bei Rückgaben) vertauscht dargestellt. Normalerweise sollte das automatisch erkannt werden. Ist dies nicht der Fall, so kann über die Mappingtabelle bei den GVCs definiert werden, ob eine Umkehrung erforderlich ist.

   Ergänzen Sie in diesem Fall die Einträge in der Mappingtabelle um den zusätzlichen Wert 1.

5) In den Sammlerdateien im Format Camt.054 können umfangreiche Informationen geliefert werden. Dies trifft insbesondere auf Lastschriftrückgaben zu.

   In der DTI-Datei stehen für den Verwendungszweck 13 Erweiterungsteile zu je 27 Zeichen zur Verfügung. Diese reichen möglicherweise nicht aus, um alle gelieferten Informationen darzustellen.

   Geben sie bitte in diesem Feld an, wie viele Zeilen für die einzelnen Informationen verwendet werden. Wenn Sie hier den Wert 0 angeben, so wird die entsprechende Information überhaupt nicht ausgegeben.

   Der hier angegebene Wert entspricht dem gewünschten Maximalwert. Stehen weniger Informationen zur Verfügung, so werden eben auch nur diese ausgegeben.

   Wenn die hier angegebenen Werte sehr hoch sind, kann es vorkommen, dass die maximale Anzahl der Erweiterungsteile überschritten wird. In diesem Fall gehen dann Informationen verloren.

   Die folgenden int-Werte sind analog zu interpretieren.

6) Wenn Sie für die zu erzeugende DTI-Datei einen Dateinamen vergeben, so wird dieser mit einer 4stelligen Erweiterung für die Dateinamen verwendet. Die Erweiterung ist erforderlich, da in einer gepackten (ZIP) Camt.054-Datei sich mehrere physikalische Dateien befinden können. Also werden die Ergebnisdateien durchnummeriert.

   Wenn Sie in diesem Feld den Wert 1 übergeben (anstatt 0), wird für die Ergebnisdatei (DTI) der Dateiname der Camt-Datei mit der Dateinamensergänzung *.dti verwendet.

   Als Pfad wird der Pfad verwendet, den Sie in der Struktur übergeben haben.

7) Das DTI-Format wurde für den deutschen Zahlungsverkehr entwickelt.

   Beim Einlesen einer Camt.054-Datei können in dieser Datei auch IBANs und BICs von anderen europäischen Ländern enthalten sein. Aus diesen können aber weder eine deutsche BLZ noch eine deutsche Kontonummer ermittelt werden.

   Auf Grund dessen kommt das zu fehlerhaften Einträgen in der DTI-Datei.

   Sie können über diesen Wert optional eine Bankleitzahl übergeben, die immer dann verwendet wird, wenn die automatisch aus der IBAN ermittelte Bankleitzahl nicht gültig ist (z.B. nicht deutsche IBAN).

8) Hier gilt das oben gesagte auch für die Kontonummer. Der hier übergebene Wert greift nur dann, wenn die ermittelte Kontonummer (möglicherweise aus einer nicht deutschen IBAN) den numerischen Wert 0 ergibt.

9) Bei der Erzeugung von Camt-054-Dateien durch die Banken, werden die Beteiligten Creditor und Debitor je nach Geschäftsvorfall unterschiedlich belegt. Bei Rücklasten z.B. sind Creditor und Debitor im Vergleich zu einer normalen Lastschrift vertauscht.

   Innerhalb von SepaTools werden die möglichen Vertauschungen über unterschiedliche GVCs (Geschäftsvorfall-Codes) ermittelt. Es kann daher vorkommen, dass ein GVC u.U. nicht richtig zugeordnet werden kann.

Wenn Sie diese Option markieren, so erfolgt keine Filterung, bzw. Prüfung der GVCs mehr. Die Zuordnung von Creditor und Debitor innerhalb der DTI-Datei erfolgt dann nur noch über die Kennzeichnung Credit oder Debit (Haben oder Soll).

10 Bei der Umsetzung von Camt.054 Dateien werden manche Informationen unstrukturiert in den Feldern "AddtlTxInf" und "AddtlNtryInf" geliefert. Gerade in der Schweiz wird hier häufig in einem dieser Felder die Referenznummer von ESR-Zahlungen geliefert. Alternativ käme für diese Referenznummer auch der strukturierte Verwendungszweck in Frage.

Da in den genannten Feldern relativ viele Zeichen übertragen werden können, ist eine Umsetzung nach DTI problematisch, da hier der Verwendungszweck aus einzelnen Zeilen mit max. 27 Zeichen besteht.

SepaTools bietet daher die Möglichkeit diese Zeichenkette nach einem bestimmten Begriff zu parsen, um so z.B. eine Referenznummer zu finden. Dazu stehen folgende, zusätzliche Felder zur Verfügung:

**RefSearch** Geben Sie bitte hier optional einen Suchbegriff an, der den Beginn der Referenznummer definiert. Z.B. Ref.-Nr. Wird dieser Begriff gefunden, so werden alle Zeichen vor diesem Suchbegriff entfernt und die Referenznummer kann ausgewertet werden.

Wenn kein Suchbegriff übergeben wird, so erfolgt auch keine Auswertung.

***RecSearchInclude*** Wird dieses Feld mit dem Wert=1 belegt, so wird in der Verwendungszweckzeile der gefundenen Referenz der Suchbegriff vorangestellt (z.B. Ref.-Nr.).

***RefSearchLen*** Geben Sie hier die maximale Länge der gewünschten Referenz an. Über diese Angabe wird zusammen mit der Position des gefundenen Suchbegriffs die Referenz ermittelt.

Die Referenz wird dann in eine der vorhandenen Verwendungszweckzeilen geschrieben (führende und folgende Leerzeichen werden entfernt).

***RefSearchOnlyTx*** Normalerweise erfolgt die Auswertung in folgender Reihenfolge:

- Zuerst wird geprüft, ob das Feld "AddtlTxInf" einen Inhalt hat.
- Ist das nicht der Fall, so wird das Feld "AddtlNtryInf" herangezogen.
- Anschließend wird geprüft, ob der Suchbegriff in "AddtlTxInf" gefunden wird.
- Wird dieser dort nicht gefunden, so wird im Feld "AddtlNtryInf" nach dem Suchbegriff gesucht.

Wird dieses Feld mit dem Wert=1 belegt, so wird ausschließlich im Feld "AddtlTxInf" gesucht.

***RefSearchAllways*** Wird dieses Feld mit dem Wert=1 belegt, so wird immer der komplette Inhalt des Informationsfeldes (max. 200 Zeichen von "AddtlTxInf" oder "AddtlNtryInf") übernommen und in den Verwendungszweckzeilen ausgegeben.

11) Beim strukturierten Verwendungszweck wird normalerweise eine Referenznummer angegeben. Dazu kann es eine Erläuterung als Code oder als Text geben.

Z.B. ISR Reference.

Ist dieses Feld mit dem Wert 1 belegt, so wird dieser Code oder Text der eigentlichen Referenznummer vorangestellt.

12) Manche Banken geben Gebühren für Lastschriftrückgaben sehr detailliert an. Dabei wird häufig zwischen eigenen und fremden gebühren unterschieden. Wenn dieses Feld mit dem Wert 1 belegt wird, so werden in der DTI-Datei unter dem Schlüssel COAM+ bis zu drei Einzelgebühren mit der Erläuterung um welche Gebühren es sich handelt, ausgegeben.

Bei Belegung des Wertes mit 0, wird nur die Summe der Gebühren ausgewiesen.

13) Manche Banken geben insbesondere bei Rücklastschriften zu Beginn des Verwendungszwecks einen festen Text aus, der für den Kunden u.U. nicht interessant ist.

Wenn dieser Führungstext nicht in die DTI-Datei übernommen werden soll, so kann hier die Startposition des gewünschten Textes innerhalb des Verwendungszwecks angegeben werden.

Lassen Sie diesen Wert bei 0, wenn der komplette Verwendungszweck übernommen werden soll.

14) Die Commerzbank hat in Teilen eine gesonderte DTI-Lösung. Grundsätzlich muss sich diese Sonderlösung nicht auswirken. Alternativ können Sie dieses Feld aber mit dem Wert **2** belegen.

### *108.4 Returncodes*

0 Alles verlief erfolgreich

-1 Die angegebene XML-Datei, bzw. ZIP-Datei konnte nicht geöffnet werden, bzw. wurde nicht gefunden.

-2 Der Pfad für die Ausgabedatei (die zu erstellende XML-Datei) ist formal ungültig (kleiner als zwei Zeichen).

-3 Die Pfadangabe (mit Dateinamen) für die Ausgabedatei (XML-Datei) ist zu lang (länger als 200 Zeichen).

-4 Das Laufwerk im Pfad für die Ausgabedatei ist nicht bereit.

-5 Das Verzeichnis, das für die Ausgabedatei angegeben wurde existiert nicht.

-6 Der Datenträger, der für die Ausgabedatei angegeben wurde ist schreibgeschützt.

-8 Die angegebene XML-Datei konnte nicht geöffnet werden.

-9 Die angegebene XML-Datei entspricht keinem gültigen Format.

-11 Innerhalb der entpackten XML-Dateien (aus der ZIP-Datei) konnte eine Datei nicht geöffnet werden.

-12 Innerhalb der ZIP-Datei mit Camt-Informationen war eine Datei enthalten, die nicht den Camt-Formaten entspricht.

-21 Die angegebene ZIP-Datei wurde nicht gefunden.

-22 Die Struktur der ZIP-Datei ist korrupt (zerstört). Die Datei kann nicht verwendet werden.

-23 Die Ausgabedateien konnten nicht erzeugt werden. Bitte prüfen Sie den Pfad, den Sie in der Funktion SepaTools_Init für das Temp-Verzeichnis übergeben haben.

-24 Sonstiger Fehler beim Entpacken der ZIP-Datei.

-25 Sie haben für die Datei DUNZIP32.DLL einen Pfad übergeben. Die Datei ist aber in diesem Pfad nicht vorhanden.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-991 Der A-Satz konnte nicht in die DTI-Datei geschrieben werden.

-992 Der C-Satz konnte nicht in die DTI-Datei geschrieben werden.

-993 Der E-Satz konnte nicht in die DTI-Datei geschrieben werden.

-994 Die DTI-Datei konnte nicht erzeugt werden.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 109 SepaTools_ConvertCamt53ToMT940

## *109.1 Zweck der Funktion*

Die Funktion liest die Camt.053-Datei aus und konvertiert die Datensätze in eine MT940-Datei.

## *109.2 Funktionsaufruf*

**int SepaTools_ConvertCamt53ToMT940(const CamtMT940Struct *CamtMT940)**

## *109.3 Parameter*

***CamtMT940*** Über diesen Parameter werden über einen Pointer auf einen Speicherbereich alle erforderlichen Variablen an die Funktion übergeben.

Die Struktur ist im Folgenden dargestellt.

struct CamtMT940Struct

| | | | |
|---|---|---|---|
| { | char | InputFile[255+1] | Der Pfad und der Dateiname für die Camt-, oder ZIP-Datei |
| | char | OutputFile[255+1] | Der Pfad und der Dateiname für die zu erzeug. DTI-Datei [1)] |
| | char | LibPath[200+1] | Optionaler Pfad für die Datei DUNZIP32.dll [2)] |
| | int | ZIP | Wert=0 wenn XML-Datei, Wert=1 wenn ZIP-Datei [3)] |
| | int | Sammler | Wert=1 für Sammlerauflösung, ansonsten Wert=0 [4)] |
| | int | OneFile | Wert=1 für „alles in eine Datei“, ansonsten Wert=0 [5)] |
| | int | BIC | BIC in der MT940-Datei darstellen [6)] |
| | int | IBAN | IBAN in der MT940-Datei darstellen |
| | int | NameAuftrag | Auftraggeber der Zahlung in der MT940-Datei darstellen |
| | int | KREF | Kundenreferenz in der MT940-Datei darstellen |
| | int | EREF | End-To-End Referenz in der MT940-Datei darstellen |
| | int | MREF | Mandatsreferenz in der MT940-Datei darstellen |
| | int | CRED | Creditor Identifikaction in der MT940-Datei darstellen |
| | int | ABWA | Abweichenden Auftraggeber in der MT940-Datei darstellen |
| | int | ABWE | Abweichenden Empfänger in der MT940-Datei darstellen |
| | int | Auto | Automatische Dateinamensbildung [7)] |
| | int | Zwischensaldo | Zwischensaldo nach jedem Umsatz [10)] |
| | char | FailBLZ[15+1] | Ersatz-Bankleitzahl [8)] |
| | char | FailKonto[15+1] | Ersatz-Kontonummer [9)] |
| | char | Reserve[500] | Reserve zur späteren Verwendung. |
| } | | | |

Zusätzliche Erklärungen:

1) Der hier vergebene Dateiname wird intern automatisch mit einer laufenden Nummer ergänzt. Dies ist erforderlich, da sich in einer von der Bank gelieferten Datei (ZIP) mehrere unterschiedliche Camt-Dateien enthalten sein können. Für jede dieser Dateien wird eine eigene Ausgabedatei erzeugt.

2) Wenn Sie mit ZIP-Dateien arbeiten wollen (die wiederum die Camt-Dateien enthalten), so ist hierfür die Datei DUNZIP32.DLL erforderlich. Diese muss auf dem Computer gefunden werden. Entweder Sie liegt im aktuellen Verzeichnis, oder die Datei wird über einen Pfad gefunden.

Wenn Sie diese DLL woanders ablegen wollen, so können Sie hier optional einen Pfad für diese DLL übergeben. Die Datei wird dann in diesem Pfad gesucht.

3) Von den Bankprogrammen werden die Camt-Formate in der Regel als ZIP-Datei mit mehreren Einzeldateien geliefert. Die ZIP-Datei muss entpackt werden, um die einzelnen Nachrichten dann einzulesen. Auch dies kann automatisiert mit SepaTools erfolgen.

   Wenn mehrere Camt-Dateien in einer ZIP-Datei enthalten sind, so werden in die gleiche CSV-Datei die Umsätze unterschiedlicher Konten und u.U. mehrerer Tage geschrieben. Über die einzelnen Felder können die einzelnen Umsätze aber genau zugeordnet werden. Es die gleiche Vorgehensweise, wie bisher bei den MT940-Dateien.

   Belegen Sie bitte dieses Feld mit dem Wert 1, wenn es sich um eine ZIP-Datei handelt.

   Wird dieses Feld mit 0 belegt, so erfolgt trotzdem eine Prüfung (über die Signatur in der Datei), ob es sich um eine ZIP-Datei handelt. Ist dies der Fall, so wird der Wert dieses Feldes intern auf 1 gesetzt.

4) Je nach Steuerung in der Bank können Sammlerauflösungen (die einzelnen Buchungen eines Eingereichten Sammlers) auch in einer Camt.053-Datei dargestellt werden. Wenn Sie diesen Wertmit ***1*** belegen, so werden die Einzelumsätze in der MT940-Datei dargestellt.

   Ansonsten wird nur die tatsächliche Sammelbuchung in die MT940-Datei übernommen.

5) Camt.053-Dateien werden immer als einzelne Dateien für ein Konto und einen Buchungstag ausgegeben. Mehrere Buchungstage oder unterschiedliche Konten werden als einzelne Dateien in einer ZIP-Datei zusammengefasst.

   Im MT940-Format ist es möglich die Kontoumsätze von unterschiedlichen Buchungstagen und Konten darzustellen.

   Wenn Sie diesen Wert mit 1 belegen, so werden die unterschiedlichen Buchungstage und/oder Konten in nur einer Datei dargestellt.

   Ansonsten wird für jeden Buchungstag und für jedes Konto eine eigene MT940-Datei erstellt.

6) In den Kontoauszügen im Format Camt.053 können umfangreiche Informationen geliefert werden.

   In der MT940-Datei stehen für diese Informationen maximal 6 Zeilen mit maximal je 65 Zeichen zur Verfügung. Diese reichen möglicherweise nicht aus, um alle gelieferten Informationen darzustellen.

   Geben sie bitte durch die Übergabe des Wertes ***1*** an, dass diese Information (in diesem Fall der BIC) verwendet werden soll.

   Wird hier der Wert ***0*** übergeben, so wird der BIC nicht verwendet.

7) Wenn Sie für die zu erzeugende MT940-Datei einen Dateinamen (Output-File) vergeben, so wird dieser mit einer 4stelligen Erweiterung für die Dateinamen verwendet. Die Erweiterung ist erforderlich, da in einer gepackten (ZIP) Camt.053-Datei sich mehrere physikalische Dateien befinden können. Also werden die Ergebnisdateien durchnummeriert.

Wenn Sie hier den Wert 1 übergeben (anstatt 0), wird für die Ergebnisdatei (MT940) der Dateiname der Camt-Datei mit der Dateinamensergänzung *.sta verwendet.

Als Pfad wird der Pfad verwendet, den Sie in der Struktur übergeben haben.

8) Das MT940-Format wurde für den deutschen Zahlungsverkehr entwickelt.

Beim Einlesen einer Camt.053-Datei werden aus der IBAN die BLZ und die Kontonummer des Kontoinhabers ermittelt. Bei einer ausländischen IBAN wird zur Identifizierung des Kontoinhabers die IBAN verwendet.

Kann auch keine IBAN ermittelt werden, so wird die in diesem Feld hinterlegte Bankleitzahl verwendet. Ist in diesem Fall auch das Feld nicht belegt, so wird die BLZ 99999999 verwendet.

9) Hier gilt das oben gesagte auch für die Kontonummer. Der hier übergebene Wert greift nur dann, wenn die ermittelte Kontonummer (möglicherweise aus einer nicht deutschen IBAN) den numerischen Wert 0 ergibt.

Ist in diesem Fall auch dieses Feld nicht belegt, so wird für die Kontonummer der Wert 1111111111 verwendet.

10) Wenn Sie dieses Feld mit dem Wert 1 belegen, so werden auch innerhalb eines Buchungstages für jeden Umsatz Zwischensalden gebildet.

### *109.4 Returncodes*

0 Alles verlief erfolgreich

-1 Die angegebene XML-Datei, bzw. ZIP-Datei konnte nicht geöffnet werden, bzw. wurde nicht gefunden.

-2 Der Pfad für die Ausgabedatei (die zu erstellende XML-Datei) ist formal ungültig (kleiner als zwei Zeichen).

-3 Die Pfadangabe (mit Dateinamen) für die Ausgabedatei (XML-Datei) ist zu lang (länger als 200 Zeichen).

-4 Das Laufwerk im Pfad für die Ausgabedatei ist nicht bereit.

-5 Das Verzeichnis, das für die Ausgabedatei angegeben wurde existiert nicht.

-6 Der Datenträger, der für die Ausgabedatei angegeben wurde ist schreibgeschützt.

-8 Die angegebene XML-Datei konnte nicht geöffnet werden.

-9 Die angegebene XML-Datei entspricht keinem gültigen Format.

-11 Innerhalb der entpackten XML-Dateien (aus der ZIP-Datei) konnte eine Datei nicht geöffnet werden.

-12 Innerhalb der ZIP-Datei mit Camt-Informationen war eine Datei enthalten, die nicht den Camt-Formaten entspricht.

-21 Die angegebene ZIP-Datei wurde nicht gefunden.

-22 Die Struktur der ZIP-Datei ist korrupt (zerstört). Die Datei kann nicht verwendet werden.

-23 Die Ausgabedateien konnten nicht erzeugt werden. Bitte prüfen Sie den Pfad, den Sie in der Funktion SepaTools_Init für das Temp-Verzeichnis übergeben haben.

-24 Sonstiger Fehler beim Entpacken der ZIP-Datei.

-25 Sie haben für die Datei DUNZIP32.DLL einen Pfad übergeben. Die Datei ist aber in diesem Pfad nicht vorhanden.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-992 Schreibfehler beim Schreiben in die MT940-Datei.

-994 Die MT940-Datei konnte nicht erzeugt werden.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.

# 110 SepaTools_ConvertCamt53ToMT940Generic

## *110.1 Zweck der Funktion*

Diese alternative Funktion liest ebenfalls die Camt.053-Datei aus und konvertiert die Datensätze in eine MT940-Datei.

## *110.2 Funktionsaufruf*

```
int SepaTools_ConvertCamt53ToMT940Generic(const char *InputFile,
                                          const char *OutputFile,
                                          const char *LibPath,
                                          int ZIP,
                                          int Sammler,
                                          int OneFile,
                                          int BIC,
                                          int IBAN,
                                          int NameAuftrag,
                                          int KREF,
                                          int EREF,
                                          int MREF,
                                          int CRED,
                                          int AbwAuftraggeber,
                                          int AbwEmpfaenger,
                                          int Auto,
                                          int Zwischensaldo,
                                          const char *FailBLZ,
                                          const char *FailKonto):Integer;
```

## *110.3 Parameter*

Die Parameter werden über einen Pointer auf alle erforderlichen Variablen an die Funktion übergeben.

Die Variablen sind ist im Folgenden dargestellt.

| | | |
|---|---|---|
| char | InputFile[255+1] | Der Pfad und der Dateiname für die Camt-, oder ZIP-Datei |
| char | OutputFile[255+1] | Der Pfad und der Dateiname für die zu erzeug. DTI-Datei [1)] |
| char | LibPath[200+1] | Optionaler Pfad für die Datei DUNZIP32.dll [2)] |
| int | ZIP | Wert=0 wenn XML-Datei, Wert=1 wenn ZIP-Datei [3)] |
| int | Sammler | Wert=1 für Sammlerauflösung, ansonsten Wert=0 [4)] |
| int | OneFile | Wert=1 für „alles in eine Datei“, ansonsten Wert=0 [5)] |
| int | BIC | BIC in der MT940-Datei darstellen [6)] |
| int | IBAN | IBAN in der MT940-Datei darstellen |
| int | NameAuftrag | Auftraggeber der Zahlung in der MT940-Datei darstellen |
| int | KREF | Kundenreferenz in der MT940-Datei darstellen |
| int | EREF | End-To-End Referenz in der MT940-Datei darstellen |
| int | MREF | Mandatsreferenz in der MT940-Datei darstellen |
| int | CRED | Creditor Identifikaction in der MT940-Datei darstellen |
| int | ABWA | Abweichenden Auftraggeber in der MT940-Datei darstellen |
| int | ABWE | Abweichenden Empfänger in der MT940-Datei darstellen |
| int | Auto | Automatische Dateinamensbildung [7)] |

| | | |
|---|---|---|
| int | Zwischensaldo | Zwischensaldo nach jedem Umsatz [10)] |
| char | FailBLZ[15+1] | Ersatz-Bankleitzahl [8)] |
| char | FailKonto[15+1] | Ersatz-Kontonummer [9)] |

Zusätzliche Erklärungen:

1) Der hier vergebene Dateiname wird intern automatisch mit einer laufenden Nummer ergänzt. Dies ist erforderlich, da sich in einer von der Bank gelieferten Datei (ZIP) mehrere unterschiedliche Camt-Dateien enthalten sein können. Für jede dieser Dateien wird eine eigene Ausgabedatei erzeugt.

2) Wenn Sie mit ZIP-Dateien arbeiten wollen (die wiederum die Camt-Dateien enthalten), so ist hierfür die Datei DUNZIP32.DLL erforderlich. Diese muss auf dem Computer gefunden werden. Entweder Sie liegt im aktuellen Verzeichnis, oder die Datei wird über einen Pfad gefunden.

   Wenn Sie diese DLL woanders ablegen wollen, so können Sie hier optional einen Pfad für diese DLL übergeben. Die Datei wird dann in diesem Pfad gesucht.

3) Von den Bankprogrammen werden die Camt-Formate in der Regel als ZIP-Datei mit mehreren Einzeldateien geliefert. Die ZIP-Datei muss entpackt werden, um die einzelnen Nachrichten dann einzulesen. Auch dies kann automatisiert mit SepaTools erfolgen.

   Wenn mehrere Camt-Dateien in einer ZIP-Datei enthalten sind, so werden in die gleiche CSV-Datei die Umsätze unterschiedlicher Konten und u.U. mehrerer Tage geschrieben. Über die einzelnen Felder können die einzelnen Umsätze aber genau zugeordnet werden. Es die gleiche Vorgehensweise, wie bisher bei den MT940-Dateien.

   Belegen Sie bitte dieses Feld mit dem Wert 1, wenn es sich um eine ZIP-Datei handelt.

   Wird dieses Feld mit 0 belegt, so erfolgt trotzdem eine Prüfung (über die Signatur in der Datei), ob es sich um eine ZIP-Datei handelt. Ist dies der Fall, so wird der Wert dieses Feldes intern auf 1 gesetzt.

4) Je nach Steuerung in der Bank können Sammlerauflösungen (die einzelnen Buchungen eines Eingereichten Sammlers) auch in einer Camt.053-Datei dargestellt werden. Wenn Sie diesen Wertmit ***1*** belegen, so werden die Einzelumsätze in der MT940-Datei dargestellt.

   Ansonsten wird nur die tatsächliche Sammelbuchung in die MT940-Datei übernommen.

5) Camt.053-Dateien werden immer als einzelne Dateien für ein Konto und einen Buchungstag ausgegeben. Mehrere Buchungstage oder unterschiedliche Konten werden als einzelne Dateien in einer ZIP-Datei zusammengefasst.

   Im MT940-Format ist es möglich die Kontoumsätze von unterschiedlichen Buchungstagen und Konten darzustellen.

   Wenn Sie diesen Wert mit 1 belegen, so werden die unterschiedlichen Buchungstage und/oder Konten in nur einer Datei dargestellt.

   Ansonsten wird für jeden Buchungstag und für jedes Konto eine eigene MT940-Datei erstellt.

6) In den Kontoauszügen im Format Camt.053 können umfangreiche Informationen geliefert werden.

In der MT940-Datei stehen für diese Informationen maximal 6 Zeilen mit maximal je 65 Zeichen zur Verfügung. Diese reichen möglicherweise nicht aus, um alle gelieferten Informationen darzustellen.

Geben sie bitte durch die Übergabe des Wertes ***1*** an, dass diese Information (in diesem Fall der BIC) verwendet werden soll.

Wird hier der Wert ***0*** übergeben, so wird der BIC nicht verwendet.

7) Wenn Sie für die zu erzeugende MT940-Datei einen Dateinamen (Output-File) vergeben, so wird dieser mit einer 4stelligen Erweiterung für die Dateinamen verwendet. Die Erweiterung ist erforderlich, da in einer gepackten (ZIP) Camt.053-Datei sich mehrere physikalische Dateien befinden können. Also werden die Ergebnisdateien durchnummeriert.

   Wenn Sie hier den Wert 1 übergeben (anstatt 0), wird für die Ergebnisdatei (MT940) der Dateiname der Camt-Datei mit der Dateinamensergänzung *.sta verwendet.

   Als Pfad wird der Pfad verwendet, den Sie in der Struktur übergeben haben.

8) Das MT940-Format wurde für den deutschen Zahlungsverkehr entwickelt.

   Beim Einlesen einer Camt.053-Datei werden aus der IBAN die BLZ und die Kontonummer des Kontoinhabers ermittelt. Bei einer ausländischen IBAN wird zur Identifizierung des Kontoinhabers die IBAN verwendet.

   Kann auch keine IBAN ermittelt werden, so wird die in diesem Feld hinterlegte Bankleitzahl verwendet. Ist in diesem Fall auch das Feld nicht belegt, so wird die BLZ 99999999 verwendet.

9) Hier gilt das oben gesagte auch für die Kontonummer. Der hier übergebene Wert greift nur dann, wenn die ermittelte Kontonummer (möglicherweise aus einer nicht deutschen IBAN) den numerischen Wert 0 ergibt.

   Ist in diesem Fall auch dieses Feld nicht belegt, so wird für die Kontonummer der Wert 1111111111 verwendet.

10) Wenn Sie dieses Feld mit dem Wert 1 belegen, so werden auch innerhalb eines Buchungstages für jeden Umsatz Zwischensalden gebildet.

### *110.4 Returncodes*

0 Alles verlief erfolgreich

-1 Die angegebene XML-Datei, bzw. ZIP-Datei konnte nicht geöffnet werden, bzw. wurde nicht gefunden.

-2 Der Pfad für die Ausgabedatei (die zu erstellende XML-Datei) ist formal ungültig (kleiner als zwei Zeichen).

-3 Die Pfadangabe (mit Dateinamen) für die Ausgabedatei (XML-Datei) ist zu lang (länger als 200 Zeichen).

-4 Das Laufwerk im Pfad für die Ausgabedatei ist nicht bereit.

-5 Das Verzeichnis, das für die Ausgabedatei angegeben wurde existiert nicht.

-6 Der Datenträger, der für die Ausgabedatei angegeben wurde ist schreibgeschützt.

-8 Die angegebene XML-Datei konnte nicht geöffnet werden.

-9 Die angegebene XML-Datei entspricht keinem gültigen Format.

-11 Innerhalb der entpackten XML-Dateien (aus der ZIP-Datei) konnte eine Datei nicht geöffnet werden.

-12 Innerhalb der ZIP-Datei mit Camt-Informationen war eine Datei enthalten, die nicht den Camt-Formaten entspricht.

-21 Die angegebene ZIP-Datei wurde nicht gefunden.

-22 Die Struktur der ZIP-Datei ist korrupt (zerstört). Die Datei kann nicht verwendet werden.

-23 Die Ausgabedateien konnten nicht erzeugt werden. Bitte prüfen Sie den Pfad, den Sie in der Funktion SepaTools_Init für das Temp-Verzeichnis übergeben haben.

-24 Sonstiger Fehler beim Entpacken der ZIP-Datei.

-25 Sie haben für die Datei DUNZIP32.DLL einen Pfad übergeben. Die Datei ist aber in diesem Pfad nicht vorhanden.

-99 Der Testzeitraum für diese Funktion ist abgelaufen. Bitte erwerben Sie eine gültige Lizenz.

-992 Schreibfehler beim Schreiben in die MT940-Datei.

-994 Die MT940-Datei konnte nicht erzeugt werden.

-999 Die API wurde noch nicht initialisiert. Bitte rufen Sie zuerst die Funktion SepaTools_Init auf.