SepaTools

DLL-Version

API-documentation

84307 Eggenfelden, 11. May 2025

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

Content

1 History of the versions 12

2 Basics 18
2.1 Data structure 18
2.2 Return codes 18
2.3 DLL-Declaration 18
2.4 Debug-version 19
2.5 Database (Datenbank.zip) 19
2.6 Database small (Datenbank_Mini.zip) 19
2.7 Functions for database small (Databank_Mini.zip) 20
2.8 Database_Big (Datenbank_Big.zip) 20
2.9 Special characters (umlauts) in SEPA XML-files 20

3 Differences and notes for the 64-bit version 21
3.1 Filenames 21
3.2 Data structure 21
3.3 Linking the DLL 21
3.4 Databases 21
3.5 SepaTools_Init 21
3.6 Function call 21
3.7 Parameters 22
3.8 Return codes 22
3.9 Versioning 22
3.10 Performance tests 22

4 Bank soft skills 23

5 Testprogram DLLDemo.exe 26

6 SepaTools_Version 27
6.1 Purpose of the function 27
6.2 Function call 27
6.3 Parameters 27
6.4 Return codes 27

7 SepaTools_SetLogFile 28
7.1 Purpose of the function 28
7.2 Function call 28
7.3 Parameters 28
7.4 Return codes 30
7.5 Example for a small Log File 30

8 SepaTools_SetLogFileInhalt 31
8.1 Purpose of the function 31
8.2 Function call 31
8.3 Parameters 31
8.4 Return codes 31

9 SepaTools_Init 32
9.1 Purpose of the function 32

9.2 Function call 32
9.3 Parameters 32
9.4 Return codes 33

10 SepaTools_Free 34
10.1 Purpose of the function 34
10.2 Function call 34
10.3 Parameter 34
10.4 Return codes 34

11 SepaTools_SetOptions 35
11.1 Purpose of the function 35
11.2 Function call 35
11.3 Parameters 35

12 SepaTools_Info 39
12.1 Purpose of the function 39
12.2 Function call 39
12.3 Parameters 39
12.4 Return codes 39

13 Create XML files 41

14 SepaTools_CreateXML 42
14.1 Purpose of the function 42
14.2 Function call 42
14.3 Parameter 42
14.4 Return codes 43

15 SepaTools_CreateXMLExt 44
15.1 Purpose of the function 44
15.2 Function call 44
15.3 Parameter 44
15.4 Return codes 45

16 SepaTools_WriteXML 47
16.1 Purpose of the function 47
16.2 Function call 47
16.3 Parameter 47
16.4 Return codes 49

17 SepaTools_WriteXMLExt 52
17.1 Purpose of the function 52
17.2 Function call 52
17.3 Parameter 52
17.4 Returncodes (additionally) 58

18 SepaTools_CloseXML 59
18.1 Purpose of the function 59
18.2 Function Call 59
18.3 Parameters 59
18.4 Return codes 59

19 XML-support for Switzerland 61
19.1 Credit Transfer 61
19.1.1 Domestic payments 62
19.1.2 Foreign Payments 63
19.2 Direct Debit 64
19.3 Plausibility Checks 65
19.4 Validation of the XML-Files 65
19.5 Implementation 65

20 SepaTools_XMLSetId 66
20.1 Purpose of the function 66
20.2 Function call 66
20.3 Parameters 66

21 SepaTools_XMLGetData 68
21.1 Purpose of the function 68
21.2 Function Call 68
21.3 Parameters 68
21.4 Return codes 69

22 SepaTools_XMLFreeData 70
22.1 Purpose of the function 70
22.2 Function call 70
22.3 Parameters 70
22.4 Return codes 70

23 To convert DTA-files in XML-files 71

24 SepaTools_ReadDTA 73
24.1 Purpose of the function 73
24.2 Function call 73
24.3 Parameters 73
24.4 Return codes 74

25 SepaTools_GetDTAFehler 75
25.1 Purpose of the function 75
25.2 Function call 75
25.3 Parameters 75
25.4 Return codes 76

26 SepaTools_GetDTAInfo 77
26.1 Purpose of the function 77
26.2 Function call 77
26.3 Parameters 77
26.4 Return codes 78

27 SepaTools_PutDTAInfo 79
27.1 Purpose of the function 79
27.2 Function call 79
27.3 Parameters 79
27.4 Return codes 82

28 SepaTools_WriteDTAtoXML 83

28.1 Purpose of the function 83
28.2 Function call 83
28.3 Parameters 83
28.4 Return codes 84

29 SepaTools_GetDTAProtokoll 85
29.1 Purpose of the function 85
29.2 Function call 85
29.3 Parameters 85
29.4 Detail Error Codes 86
29.5 Return codes 87

30 SepaTools_FreeDTAVars 88
30.1 Purpose of the function 88
30.2 Parameters 88
30.3 Return codes 88
30.4 Flow chart 89
30.5 Description of the flow chart 91

31 Convert DTAZV-files in XML-files 93

32 SepaTools_ConvertDTAZVtoXML 94
32.1 Purpose of the function 94
32.2 Call of the function 94
32.3 Parameters 94
32.4 Returncodes 95

33 SepaTools_GetDTAProtokoll 96
33.1 Purpose of the function 96
33.2 call of the function 96
33.3 Parameters 96
33.4 Detail-Error-codes 96
33.5 Returncodes 96

34 To convert CSV-files into XML-files 96

35 SepaTools_ConvertCSVtoXML 98
35.1 Purpose of the function 98
35.2 Call of the function 98
35.3 Parameters 98
35.4 Returncodes 102

36 SepaTools_GetDTAProtokoll 104
36.1 Purpose of the function 104
36.2 Call of the function 104
36.3 Parameters 104
36.4 Detail-Error-codes 104
36.5 Returncodes 104
36.6 Example for CSV-Converting 104
36.6.1 Set of the structure 105

37 To Create DTAZV files 106

38 SepaTools_CreateAZV 107
38.1 Purpose of the function 107
38.2 Function call 107
38.3 Parameters 107
38.4 Return codes 108

39 SepaTools_WriteAZV 109
39.1 Purpose of the function 109
39.2 Function call 109
39.3 Parameters 109
39.4 Return codes 110

40 SepaTools_CloseAZV 113
40.1 Purpose of the function 113
40.2 Function call 113
40.3 Parameters 113
40.4 Return codes 113

41 SepaTools_GetLandWhg 114
41.1 Purpose of the function 114
41.2 Function call 114
41.3 Parameters 114
41.4 Return codes 115

42 SepaTools_SetVersionUndLand 116
42.1 Purpose of the function 116
42.2 Function call 116
42.3 Parameters 116
42.4 Return codes 117

43 SepaTools_XMLLesenInit 119
43.1 Purpose of the function 119
43.2 Function call 119
43.3 Parameters 119
43.4 Return codes 119

44 SepaTools_XMLLesen 120
44.1 Purpose of the function 120
44.2 Function call 120
44.3 Parameters 120
44.4 Return codes 121

45 SepaTools_XMLLesenClose 123
45.1 Purpose of the function 123
45.2 Function call 123
45.3 Parameters 123
45.4 Return codes 123

46 SepaTools_GetBIC 124
46.1 Purpose of the function 124
46.2 Function call 124
46.3 Parameters 124
46.4 Return codes 124

47 SepaTools_CheckIBAN 125
47.1 Purpose of the function 125
47.2 Function call 125
47.3 Parameters 125
47.4 Return codes 125

48 SepaTools_CheckIBAN_Strong 126
48.1 Purpose of the function 126
48.2 Function call 126
48.3 Parameters 126
48.4 Return codes 126

49 SepaTools_GetBLZKonto 127
49.1 Purpose of the function 127
49.2 Call of the function 127
49.3 Parameters 127
49.4 Returncodes 128

50 SepaTools_CheckIBANLand 129
50.1 Purpose of the function 129
50.2 Function call 129
50.3 Parameters 129
50.4 Return codes 129

51 SepaTools_CheckCI 130
51.1 Purpose of the function 130
51.2 Function call 130
51.3 Parameters 130
51.4 Return codes 130
51.5 Known Length of CIs 130

52 SepaTools_CheckESR 132
52.1 Purpose of the function 132
52.2 Function call 132
52.3 Parameters 132
52.4 Return codes 132

53 SepaTools_GetESRPrZiff 133
53.1 Purpose of the function 133
53.2 Function call 133
53.3 Parameters 133
53.4 Return codes 133

54 SepaTools_CheckSEPATeilnahme 134
54.1 Purpose of the function 134
54.2 Function call 134
54.3 Parameters 134
54.4 Return codes 134

55 SepaTools_ConvertBLZKonto 135
55.1 Purpose of the function 135
55.2 Function Call 135
55.3 Parameters 135

55.4 Return codes 136

56 SepaTools_SetErfahrung 138
56.1 Purpose of the function 138
56.2 Function call 138
56.3 Parameters 138
56.4 Return codes 138

57 SepaTools_GetBankInfo 139
57.1 Purpose of the function 139
57.2 Function call 139
57.3 Parameters 139
57.4 Return codes 139

58 SepaTools_GetBICInfo 141
58.1 Purpose of the function 141
58.2 Function call 141
58.3 Parameters 141
58.4 Return codes 142

59 SepaTools_GetIBANLand 143
59.1 Purpose of the function 143
59.2 Function call 143
59.3 Parameters 143
59.4 Return codes 143

60 SepaTools_GetSEPABank 144
60.1 Purpose of the function 144
60.2 Function call 144
60.3 Parameters 144
60.4 Return codes 145

61 SepaTools_GetBLZ 146
61.1 Purpose of the function 146
61.2 Function Call 146
61.3 Parameters 146
61.4 Return codes 147

62 SepaTools_FindBLZ 148
62.1 Purpose of the function 148
62.2 Function call 148
62.3 Parameters 148
62.4 Return codes 149

63 SepaTools_SearchBLZ 150
63.1 Propose of the function 150
63.2 Function call 150
63.3 Parameters 150
63.4 Return codes 151

64 SepaTools_CheckPruefziffer 152
64.1 Purpose of the function 152
64.2 Function call 152

64.3 Parameters 152
64.4 Return codes 152

65 SepaTools_GetPurposeCode 153
65.1 Purpose of the function 153
65.2 Function call 153
65.3 Parameters 153
65.4 Return codes 154

66 SepaTools_SearchPurposeCodes 155
66.1 Purpose of the function 155
66.2 Function call 155
66.3 Parameters 155
66.4 Return codes 156

67 SepaTools_GetReasonCode 157
67.1 Purpose of the function 157
67.2 Function call 157
67.3 Parameters 157
67.4 Return codes 157

68 SepaTools_SearchReasonCodes 158
68.1 Purpose of the function 158
68.2 Function call 158
68.3 Parameters 158
68.4 Return codes 159

69 SepaTools_AddUserBIC 160
69.1 Purpose of the function 160
69.2 Function call 160
69.3 Parameters 160
69.4 Return codes 160

70 SepaTools_DelUserBIC 161
70.1 Purpose of the function 161
70.2 Function call 161
70.3 Parameters 161
70.4 Return codes 161

71 SepaTools_GetUserBIC 162
71.1 Purpose of the function 162
71.2 Function call 162
71.3 Parameter 162
71.4 Return codes 162

72 SepaTools_FindUserBIC 163
72.1 Purpose of the function 163
72.2 Function call 163
72.3 Parameters 163
72.4 Return codes 163

73 SepaTools_AddUserIBAN 164
73.1 Purpose of the function 164

73.2 Function call 164
73.3 Parameters 164
73.4 Return codes 164

74 SepaTools_DelUserIBAN 166
74.1 Purpose of the function 166
74.2 Function call 166
74.3 Parameter 166
74.4 Return codes 166

75 SepaTools_GetUserIBAN 167
75.1 Purpose of the function 167
75.2 Function call 167
75.3 Parameter 167
75.4 Return codes 167

76 SepaTools_FindUserIBAN 168
76.1 Purpose of the function 168
76.2 Function call 168
76.3 Parameters 168
76.4 Return codes 168

77 SepaTools_GetTargetDatum 169
77.1 Purpose of the function 169
77.2 Function call 169
77.3 Parameter 169
77.4 Return codes 170

78 SepaTools_IsTargetDatum 171
78.1 Purpose of the function 171
78.2 Function call 171
78.3 Parameters 171
78.4 Return codes 171

79 Create pain.007 files 172

80 SepaTools_CreateXML007 172
80.1 Purpose of the function 172
80.2 Function call 172
80.3 Parameter 172
80.4 Return codes 173

81 SepaTools_WriteXML007 175
81.1 Purpose of the function 175
81.2 Function call 175
81.3 Parameter 175
81.4 Return codes 176

82 SepaTools_CloseXML007 178
82.1 Purpose of the function 178
82.2 Function Call 178
82.3 Parameters 178
82.4 Return codes 178

83 Create CGI files 180

84 SepaTools_CreateCGI 181
84.1 Purpose of the function 181
84.2 Function call 181
84.3 Parameter 181
84.4 Return codes 182

85 SepaTools_WriteCGI 183
85.1 Purpose of the function 183
85.2 Function call 183
85.3 Parameter 183
85.4 Return codes 186

86 SepaTools_CloseCGI 187
86.1 Purpose of the function 187
86.2 Function Call 188
86.3 Parameters 188
86.4 Return codes 188

1 History of the versions

In this part, the essential modifications of the documentation are updated chronologic.

Version	Date	Document modification
3.85.2	11.05.2025	There is a new function ***SetNewUltimate***. Please use it for the placement of the ultimate Debtor or Creditor. If you place the value “1” to this function, the ultimate Debtor or Creditor will be placed in the transactions area. This is useful, if you have different payer for every payment.
3.82	03.08.2023	- Please use for all addresses the „Struct“ variable.
3.80	06.05.2023	- By converting DTA files you can pass a time in the structure SepaTools_PutDTAInfo. The structure ***DTAInfoStruct*** has been extended. - By Converting CSV files you can pass the in the structure ***CSVConvertStruct***. - Some banks provide the invalid date February 30. This will be corrected in 28. or 29. February. - The databases should be changed.
3.70	09.02.2023	- The validation of MT940 files has been extended (slightly looser).
3.61	04.11.2022	- The structure XMLWriteExtStruct got a new field CHQRInfo wich was added at the last of the structure. It can hold additional information’s for the Switzerland QR invoices.
3.60	06.08.2022	- No changes.
3.50	06.05.2022	- There are the new functions SepaTools_CheckESR and SepaTools_GetESRPrZiff for the Switzerland ESR numbers. Please see the documentation.
3.40	09.02.2022	- The check method of the German check rule A4 was modified. - The return codes of the functions SepaTools_CheckIBAN and SepaTools_GetBLZKonto has been modified.
3.30	06.11.2021	- Changes are only made for Camt functions. Please see for this the German documentation.
3.20	03.08.2021	- There are the new functions ***SepaTools_GetReasonCode*** and ***SepaTools_SearchReasonCodes*** to check or obtain reason codes. - The function ***SepaTools_ConvertCSVToXML*** supports now IBAN Only, if you pass a valid IBAN. Header lines are supported also now. Please see documentation. - The function ***SepaTools_CheckIBAN*** and ***SepaTools_CheckIBAN_Strong*** provides now to additional return codes. Please see documentation. - The return codes of the function ***SepaTools_GetBLZBank*** has been expanded with two bit coded positive values. Please see documentation.

Version	Date	Document modification
3.10	05.05.2021	- The structure ***XMLWriteExtStruct*** has been expanded in the reserve area (Fields ***EmpfAdrHausnummer*** and following). - Important note for users who use Switzerland payments with the payment kind 4 or 6. The fields of the address of the bank of payment receiver have been changed. Please see the documentation.
3.00	04.02.2021	- The function ***SepaTools_GetBLZKonto*** provides no results for bank data from Poland. - The databases contain now data for the purpose codes. There are the new functions ***SepaTools_GetPurposeCode*** and ***SepaTools_SearchPurposeCodes*** provided.
2.91	04.11.2020	- The structure XMLCreateExtStruct has been modified.
2.90	04.08.2020	- For checking the IBAN there is a new function SepaTools_CheckIBAN_Strong. This function doesn't filter invalid characters from the passing IBAN. Are there invalid characters, the function returns ay negative value. Please see for the details the documentation.
2.81.0	29.04.2020	- A memory error while creating CGI files is fixed.
2.80	10.02.2020	- A minor program weakness has been eliminated when converted MT940 files. - There was a problem while creating ESR debit payments for Switzerland.
2.71	02.11.2019	- Only changes in the Camt module.
2.70	07.09.2019	- For the function ***SepaTools_SetVersionUnd Land***, the additional value 7 for the XML version 3.3 is now available. - The function ***SepaTools_GetBLZKonto*** supplies now results for Switzerland and Liechtenstein. For this reason, the structure ***BLZKontoStruct*** was modified. - The structure XMLOptionStruct was extended with two fields in the reserve area.
2.60.2	03.05.2019	- The return code -9 in the function SepaTools_GetBLZKonto was removed. - The databases should be changed.
2.60	05.02.2019	- The IBAN rule 049 was modified.
2.50	14.08.2018	- The functionality of the field ***EilUebw*** in the structure ***XMLOptionStruct*** has been extended. - The structure ***XMLOptionStruct*** as an additional field ***ISOSonder***. - In the reserved area of the structure ***XMLWriteExtStruct*** is an additional field ***AusfZeit*** placed.
2.40	06.05.2018	- The structure ***CGICreateStruct*** has been extended with the fields ***ShortMsgId*** and ***WhgSort*** (Reserved area). - The structure ***XMLWriteExtStruct*** has been extended with two fields for the Swiss payments.

Version	Date	Document modification
2.30	01.02.2018	- Only the databases are new.
2.20	31.10.2017	- The API can create pain.007 XML files. Three functions are necessary for this. This file format is used for reversal direct debit payments (e.g. double presentation). - The support of pain.007 files through your payment provider is optional. In opposite to the processing of XML payment files, there is no obligation to support these files by the banks. - The API can now create CGI files. With these files, you can create foreign payments outside of the SEPA area. Please remind that these files are currently only supported by big banks. In our case the files was validated with NORDEA bank Finland.
2.10	02.08.2017	- There is a new function ***CreateXMLExt***. This is an optional replacement for the function ***CreateXML***, because this function has no reserved area. - Since Version 3.1 it is possible to place the block ***PmtTpInf*** in the transaction level. - The structure ***XMLWriteExtStruct*** has been expanded in the reserved area. Now you can pass the address of the recipient, the initiator and a category purpose code. For Switzerland, there is the additional field ***ChAccontPrtry***. - In the function ***SepaTools_ConvertCSVtoXML***, you can now pass address fields for the initiator and the recipient. The structure ***CSVConvertStruct*** was modified. - By using the function SepaTools_ConvertDTAZVtoXML the address information of the initiator and of the recipient are also written in then XML file. - Now it is possible to sort the XML entries according to the currency. This is only useful with Switzerland payments. - Only for file presenting in Switzeland, it is possible to pass additional fields with contact information to the Group Header. Please refer to the function ***CreateXMLExt***.
2.00	07.05.2017	There is a new option for using the Slash (/) or double Slash (//) in text fields. Please see the function SepaTools_SetOptions. The check digit method of B1 was modified. The additional check digit method of E3 was included. The IBAN rule 54 would be removed.

Version	Date	Document modification
		The databases are updated.
1.90.0	03.02.2017	The additional check digit method of E3 was included.
1.80.0	02.11.2016	The check digit calculation in mode 74 was expanded. The IBAN rule no. 42 was expanded to support some 10 digits account numbers. The function ***SepaTools_CheckPruefziffer*** supports an additional return code of -5. This indicates if you pass an empty account number. The databases are updated.
1.70.0	02.08.2016	A set of four new functions was included. Witch these functions (SepaTools_CreateAZV and followings), you can create foreign payments according the German DK-documentation. See also the Website www.sepa-tools.de and here the section “Sonstige Infos”.

1.60.0	09.05.2016	- The new DK XML-version (new scheme), valid at 20. November 2016 is supported. The following are the essential changes:
		  - At this version, there are only short time direct debits (D+1).
		  - The instrument RCUR is now valid for first und for recurrent direct debit.
		  - Mandate-Ids can contain spaces.
		  - The schema for mandate amendments has changed.
		- See for further information the download area at www.sepa-tools.de

1.50.0	01.11.2015	- The presenting of xml-files in Switzerland is now supported. Because of the support to create xml-files in Switzerland, the data structure was modified and extended. Because of the differentiated control of the payment in Switzerland, this support needs more data fields in the data structure. If you don’t use the fields ***StructZweck***, ***StructTyp*** and ***BatchBooking***, you have nothing to do or to adapt.
1.49.0	18.08.2015	- In the function *SepaTools_GetVersion* one error is fixed (the integer value for the item “version” was 0 instead of 1.
1.48.0	05.05.2015	- No basic changes. For changes in the camt-module see German documentation.
1.47.0	07.02.2015	- The meaning of the parameter “Skip” in the function *SepaTools_GetTargetDatum* was expanded. - The function *SepaTools_SetLogFile* was expanded. Additional to the log-file, message dialogs can be displayed. - There is a new function *SepaTools_SetLogFileInhalt*. - Within the SEPA XML-file can special characters be included. You can control this with the function *SepaTools_SetOptions*. The structure for this function was modified.
1.46.0	17.11.2014	- There is a new function *SepaTools_SetLogFile*. Thus the sequence of the DLL-functions can be traced in a text file. - The meaning of the return code -6 of the function *SepaTools_CheckCI* was modified. - The structure *BLZKontoStruct* was expanded (modified). - The structure *DTAInfoStruct* was expanded (modified).

Version	Date	Document modification
1.45.0	27.98.2014	- The function *CheckCI* provides an additional return code -6. - The structure *XMLOptinStruct* was expanded. With the value *CompressXML* can be defined if the XML-file is compressed (without linefeeds) or not. - The structure XMLWriteExtStruct was expanded. The value *BatchBooking* can be set explicit.
1.44.1	23.05.2014	- The structure *XMLOptionStruct* was modified and expanded. - The structure *XMLWriteExtStruct* was modified and expanded.
1.44.0	08.05.2014	- The function *CheckCI* has an additional return code with the value -6. - By using the function *XMLLesen*, the Field *SammlerSumme* in the structure *XMLReadStruct* was not filled. This bug is fixed now.
1.43.1	20.02.2014	- For the function *SetOptions* are the additional variables “Sammler-Trennen” and “MaxXMLSatz” available.
1.43	05.02.2014	- There is an additional function *GetBICInfo*. You can retrieve informations of the bank routing number by passing the BIC (only from German and Austrian banks). - By using the function *GetBankInfo*, the participation of COR1 can be checked. - There is a new return-code -123. It will be provided, if the value 2 is set to the field B2B and the receiving bank does not support this. - The function *SetOptions* was expanded. - When converting CSV-files to XML-files, you can pass an individual EndToEnd-ID. This value overwrites the automatically calculated EndToEnd-ID. - The function GetBankINfo has an additional, positive return code of 1 (this is a note, no error).
1.42	06.11.2013	- The return code -115 for the function XMLWrite is obsolete. XML records without purpose can now be created.
1.41	31.08.2013	- For the functions *XMLWrite* and *XMLWriteExt* exits now the additional return codes -120 until -122. - The “Deutsche Bank” (German Bank) has changed her IBAN-rule. Affected are 7 digits and 10 digits account numbers.
1.40	04.08.2013	- The reserved area of the data structure XMLWriteExtStruct has increased to the amount of 1.500 bytes. This is for later using. - The function XMLClose returns an additional return code of -8.
1.39	29.06.2013	- Expansion of the function *SetOptions* - New and extended function *WriteXMLExt* - New function *ConvertDTAZVtoXML* - New function *ConvertCSVtoXML* - New function *GetBLZKonto* - Optional expansion of the parameter B2B in the structures *XMLWriteStruct, DTAInfoStruct* and *XMLReadStruct.*

Version	Date	Document modification
		- Within the functions *CheckSEPATeilnahme* and *GetBankInfo,* there is the additional returncode -3. - Within the function GetBIC, there is the additional returncode -2.

2 Basics

The following documentation describes a library which supports third party products by the SEPA implementation. It is a Windows-DLL for 32-bit Windows programs.

In principle the Tools was developed for the German banking industry. In far parts it also supports Austrian banking accounts. Due to the specification of a 5 digit bank identifier, it recognized an Austrian banking account.

The DLL is named SepaTools.dll.

All data will be transferred in predefined data structures (in both directions). The syntax of then programming language C is used.

2.1 Data structure

All API-functions expect the specified data structures with at least the specified size. Access to all data structures happens with pointers.

The length of char variables is always set to the maximal length of the variable +1 char for the closing NULL.

Reference:

It is the responsibility of the calling application for the right size of the handed memory area.

Structures will be up to the size of the structure filled. It is in the responsibility of the calling application to allocate the memory area.

Note:

Same data structures contain reserved areas. Please be sure, that this reserved area is filled with binary NULL values. Otherwise, you may get unexpected results and errors

All integer values have a size of 4 bytes. There are no double values in the API. Amounts will be shown in char arrays in euro cent. For example, the value 1.234,56 € corresponds to the string “123456”.

2.2 Return codes

The result of all return codes is normally a negative integer value (32 bit/4 byte). If other return codes exist, it will be explicit explained.

Return codes with the value 0 indicate everything is O.K. If there are return codes with values greater than 0 then it is a reference. The application can continue on like Return codes of 0.

2.3 DLL-Declaration

All DLL-functions are stdcall declared. Case sensitivity is to note! All data structures are by default 8 byte aligned (#pragma pack 8).

In field tests we have seen, that some compiler (partially Cobol compiler) could not work with data structures with 8 byte alignment. For this case you can use the additional DLL with a byte alignment of 1 byte.

The name of this DLL (1 byte alignment, #pragma pack 1) is **SepaTools_P1.dll**. Possibly you have to change the filename.

In additional to this DLLs there is a DLL with a byte alignment of 4 bytes (#pragma pack 4). The name of this DLL is **SepaTools_P4.dll**. Possibly you have to change the filename.

The API is developed in the programming language Delphi. You have to create the C-header files by you own.

We recommend a dynamic loading of the DLL by using the Windows function LoadLibrary and catching the function pointers with the function GetProcAddress.

2.4 Debug-version

From the Version 1.46.1, debug-versions are not available longer. Instead of this, you can use the function *SepaTools_SetLogFile*.

2.5 Database (Datenbank.zip)

To get various error checks, SepaTools needs 2 databases. It are the bank identifier file from the German National Bank (BLZ.dat and BLZ.idx) and the database with the by SEPA reachable banks with additional information (Sepa.dat and Sepa.idx).

In principle the databases (as part of the API) are expected in the actual directory. Because with the function “SepaTools_Init”, you can define a differ directory for this databases.

These databases are updated regularly (at most every three month). You find the updates at the website http://www.sepa-tool.de.

The API-DLL checks the version of the database. If the version of the database is not the same version as the version from the DLL, the function “SepaTools_Init” returns a negative value.

You should always update the API-DLL and the databases together.

The database works until now always in network mode.

2.6 Database small (Datenbank_Mini.zip)

The described databases are very extensive. The reason is, that these databases contain the BICs and addresses of all in Europe by SEPA reachable banks.

This information is not necessary for all the functions in the DLL. If you only use following functions, then you can use smaller databases. The database contains the files “Sepa_Mini.dat” and “Sepa_Mini.idx“. Please rename the files in “SepaTools.dat” and “SepaTools.idx“.

The files “BLZ.dat” and BLZ.idx” are used as usual.

2.7 Functions for database small (Databank_Mini.zip)

The following functions can be used with the small database.

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

2.8 Database_Big (Datenbank_Big.zip)

The database Sepa.dat/Sepa.idx contains all for SEPA reachable banks. Basic for this are the “SCT register of participants” of the EPC and the “SCL-directory” of Deutsche Bundesbank. The SCL-directory contains all reachable BICs.

Unfortunately, this directory does not contain the location of the bank. The search for a particular bank with the function *SepaTools_GetSEPABank* does not result the desired result (no location available).

To get a better result at last for the German banks, the database_big (consisting of the files Sepa_Big.dat and Sepa_Big.idx) was expanded with the bank locations of the BLZ-directory (Deutsche Bundesbank).

This additional database is only useful by using the function *SepaTools_GetSAEPABank*. If you like to use this database, you have to rename the files Sepa_Big.dat and Sepa_Big.idx in Sepa.dat and Sepa.idx.

Please see also the variable *NotNameOrt* in the structure *XMLOptionStruct*.

2.9 Special characters (umlauts) in SEPA XML-files

Normally it is allowed that you pass special characters into a SEPA XML-file. It is obligated for the banks to accept that special characters (i.e. ÄÖÜäöüß). However the banks can replace these special characters with other characters or blanks.

If you chose the function *SepaTools_SetOptions* to indicate that you want to pass special characters, it is not certain that these characters arrive the payee.

If you do nothing special, SepaTools will convert the characters ÄÖÜäöü? into Ae, Oe, Ue, ae, oe, ue, ss.

3 Differences and notes for the 64-bit version

The application of the 64-bit version is in principle identical to the application of the 32-bit version. There are some little differences and notes, which will described in this chapter.

If there are no notes in this chapter, the documentation of the 32-bit version is valid.

3.1 Filenames

The filename for the 64-bit version of the DLL is named ***SepaTools64.dll*** for the DLL with a byte alignment of ***8 bytes*** and ***SepaTools64P1.dll*** for the DLL with a byte alignment of ***1 byte***.

3.2 Data structure

The data structures of the 64-bit version are identical to the 32-bit version. This means, that all integer values (int) have a size of 4 byte. All chars, strings and pointer to strings contain chars with a size (length) of 1 byte. This is identical to the 32-bit version.

Note:

There are no wide chars and wide strings, which contains chars with size (length) of 2 bytes. Only AnsiChar and pointer to AnsiChar (PAnsiChar) are valid.

3.3 Linking the DLL

The DLL is not a COM-object. To link the DLL, you have to use the Windows function ***LoadLibrary***, in the same way as with the 32-bit version. You get the function pointer by using the Windows function ***GetProcAddress***.

3.4 Databases

The same compact databases as by the 32-bit version are used. Unfortunately, they can not be directly linked to the 64-bit version.

The connection to the DLL is made by using a 32-bit database server, which is named SepaDB.exe. It will be automatically called and administrated by the 64-bit DLL.

3.5 SepaTools_Init

This is the only one function call, which differs in parts of the parameters and return codes to the 32-bit version.

3.6 Function call

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

3.7 Parameters

DataPath	See 32-bit version.
TempPath	See 32-bit version.
UserPath	See 32-bit version.
Lizenz	For the 64-bit version, you need an own, from the 32-bit version different license key.
Serverpath	Please pass the path for the database server SepaDB.exe. Please pass only the path name without the filename. The filename will be automatically expanded.
Netz	this parameter is no longer required in the 64-bit version. The databases will be always opened in network mode.

3.8 Return codes

For this function, there are the following, additional return codes:

-20	The database server could not be found. Please check the path which is passed to the variable ***ServerPath***.
-21	The database server and the 64-bit DLL don't have the same version. Please download the actual versions from the website www.sepa-tool.de.
-998	The database server is not active. This can really only happen, if the database server is crashed (what not should occur) or the database server is terminated manually. **Please note:** **This return code can occur by all DLL-functions, because before ever function call there is a live check of the database server.**

3.9 Versioning

The 64-bit version gets always the same version number as the 32-bis version. This is based on the same professional contents of the two DLL-versions.

The version date can be different, because the two versions are not necessary build on the same day.

3.10 Performance tests

With the 64-bit version performance tests were made by using two notebooks. A different count of debits would be created with the function SepaTools_Write XMLExt.

In all cases BIC and IBAN was calculated by using bank routing number (BLZ) and bank account number by access to the database server. For these tests the following notebooks was used.

The result of the performance test is depends very much from the performance of the hard disk. Similarly, the other load on the computer plays a crucial role

The values given are therefore only approximate values.

	Test1	*Test2*
Model	Lenovo ThinkPad T530	Lenovo ThinkPad P140 p
Date of build	June 2013	November 2014
Hard disk	500 GB	1 TB
Operating system	Windows 7 SP 1	Windows 8.1 Pro
Main memory	8 GB	16 GB
Processor	I7-3740 QM CPU 2,7 GHz	I7-4910 QM CPU 2,9 GHz

Results of the Tests:

Debits count	*Test1 Time MM:SS*	*Test2 Time MM:SS*	*File size*	*lines (count) in the XML-file*
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 Bank soft skills

The DLL provides a technical interface for the realization of SEPA functionality. The bank-specific fundamentals are assumed. In the following, you can find the notes of essential terms and references to sources of information.

Term	Explanation
BLZ - Bank routing bumber	The national address of a bank. The BLZ has 8 digits in Germany and 5 digits in Austria. For information visit the following link. http://www.bundesbank.de/Navigation/DE/Kerngeschaeftsfelder/Unbarer_Zahlungsverkehr/Bank identifier codeen/bank identifier codeen.html
Bank account number	The national bank account number of a customer. the account number is in Germany from 2 to 10 digits. In Austria, the account number is from 2 to 11 digits.

Term	Explanation
	Within the API, the account number is always 10 digits (Germany) or 11 digits (Austria) represented.
BIC	The BIC is the international address of a bank. The BIC is 8 digits or 11 digits. Examples of valid BICs: DAAEDED1030 BFECGPGXXXX BFECGPGX At the 5th and 6th position of a BICs is the country code (ISO 3166-1), e.g. DE for Germany. For a list of ISO country codes, visit: http://www.mathguide.de/projekt/doku/landcode.html
IBAN	The IBAN is the international bank account number of a customer. The length of the IBAN varies from country to country. In Germany, the IBAN has 22 digits and 20 digits in Austria. The first two letters are the ISO country code (DE for Germany). Points 3 and 4 represent the test number (ISO 7064). Further information is available via the link: http://de.wikipedia.org/wiki/International_Bank_Account_Number
Creditor Identifica-tion (CI)	Anyone who wants to collect direct debits with SEPA needs a creditor identification (CI) from the European Central Bank. In Germany, the CI is awarded by the Deutsche Bundesbank. The length of the CI differs from country to country. In Germany, the CI has a length of 18 characters. For more information concerning the CI and their construction in Germany, see: http://www.bundesbank.de/Navigation/DE/Kerngeschaeftsfelder/Unbarer_Zahlungsverkehr/SEPA/Glaeubiger_Identifikationsnummer/glaeubiger_identifikationsnummer.html
Country Code	Country Codes have always 2 digits according to ISO 3166-1. For a list of ISO country codes, visit: http://www.mathguide.de/projekt/doku/landcode.html
SEPA Instruments	There are three different Instruments for SEPA payments: CT Credit Transfer – Transfers DD Core Direct Debit - Direct Debit. The same as the previous Direct Debit with authorization.

Term	Explanation
	B2B Direct Debit - Direct Debit Business. The same as the previous Direct Debit with debit order. Note: DD Direct Debits and B2B direct debits may not be mixed within a physical SEPA XML file, although technically possible. A mixed file would be rejected by the banking system.
Available banks by SEPA	A list of banks with are available for SEPA see: http://www.bundesbank.de/Navigation/DE/Kerngeschaeftsfelder/Unbarer_Zahlungsverkehr/SEPA/SCL_Directory/scl_directory.html
Specification of SEPA-data formats	For the technical specification of the SEPA data formats, see: http://www.ebics.de/index.php?id=77

5 Testprogram DLLDemo.exe

The test program DLLDemo.exe serves for testing the fundamental behavior of the DLL.

Here you find the test program:

http://www.sepa-tool.de (Menu item "DLL version").

The test program can call some of the basic functions of the DLL. The test program itself is of no intelligence of its own, it serves only to capture the data structures, witch have to be transferred.

All return values correspond to the return codes witch are documented here. Please use the test program only in conjunction with this documentation.

Note:

The test program DLLDemo.exe has to be in the same directory as the file SepaTools.dll.

6 SepaTools_Version

6.1 Purpose of the function

The function delivers information about the version of SepaTools DLL (API). This function can be called without previous initializing of the API.

6.2 Function call

void SepaTools_Version(XMLVersionStruct *VersionStruct)

6.3 Parameters

VersionStruct This is a pointer to the structure XMLVersionStruct. In this structure, the version information is returned.

The structure is shown below.

struct MLVersionStruct

{	char	Datum[8+1]	Versions date format DDMMYYYY.
	int	Version	Versions number
	int	SubVersion	Sub-versions number
	char	Versionsstring[20+1]	Version and sub-version as String displayed.
}			

6.4 Return codes

none.

7 SepaTools_SetLogFile

7.1 Purpose of the function

This function can be called before all other functions. A log file will be created as a text file. Within the program sequences, this function can be called once than more.

7.2 Function call

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

7.3 Parameters

Path

Please pass a pointer to the directory (including filename) in which the log file should be created.

If you pass a path or/and a filename, which contains one or more question mark (**?**), than the filename will be updated (added) with a unique number (with date and time information). the question marks will be removed from the file name.

This is useful, if you call the function SepaTools_Init multiple. Than the log file will not be overwritten.

Example:

Passed path (file name):

C:\Test\TestLogFile?.txt

Created file name (e.g.):

C:\Test\TestLogfile_20141215_155914_938528.txt

Action

Please define the action, which should be processed. The following values are available.

1 A new log file with the passed filename will be created. If the named log file already exists, it will be overwritten.

2 An existing log file will be expanded with additional records (append). If the log file with the passed filename does not exist, the function returns with an error code.

In this case, you should call prior the same function with the parameter 1 (new file).

3 The log file will be stopped. The file will be closed.

This procedure is useful, if only some sequences of the program should be traced.

4 The log file will be closed. This function call has the same meaning as a call with the value 3. The function call with the value 4 is automatically called within the function SepaTools_Free.

The difference is only informative. The Value 3 shows that the tracing is interrupted temporary. The value 4 shows that the log file is closed finally.

Inhalt With this parameter, you tell the function in which details the traces occur. The parameter is a combination of 3 Values. The values are combined bitwise.

1 All information will be traced. Information means no errors, nevertheless they documents single fields inside the DLL function.

2 This value defines to trace notes. Notes are useful, if values was changed inside the DLL functions.

4 Use this value to trace errors.

8 Use this value to display additional to the text file message dialogs. The displaying of the message dialogs can be aborted within every message dialog.

This behavior is useful, when you like to proof the parameters you have passed to the function *SepaTools_SetLogFile*. Naturally there could not be written a log-File at this time.

Note: All records will be traced, if you use the value 15 (1 + 2 + 4 + 8).

Flush This parameter defines the behavior while writing the records in the log file. You have the following selection:

0 No further action. In the case that the program will abort with an exception, some record shortly written can lose. The reason is why, that the closing of the log file occurs in the function SepaTools_Free.

1 After each record write, the log file will be flushed to disk. This saves the log file in case of an exception.

2 If you set this value, than the log file will be closed and reopened after every written to it.

If you set the value of this parameter unless 0, 1 or 2, the value is set internally to 2.

7.4 Return codes

0	Everything was successful
-1	The length of the passed parameter to ***Path*** is too short, less than 5 characters.
-2	The parameter ***Action*** is out of the range of 1 to 4.
-3	The parameter ***Inhalt*** is out of the range of 1 to 15.
-4	The drive of the passed directory is not ready.
-5	The named directory does not exist.
-6	The drive of the named directory is not ready.
-7	The log file could not be created.
-8	The log file could not be expanded (parameter Action=2)

7.5 Example for a small Log File

```
--------------------------------------------------------------------------------
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 Purpose of the function

In connection with the function SepaTools_SetLogFile, you can control the size of output of the log-file. So you can control different sizes of the log-file output (or no output) within your program flow.

8.2 Function call

int SepaTools_SetLogFileInhalt(int Inhalt)

8.3 Parameters

Inhalt With this parameter, you tell the function in which details the traces occur. The parameter is a combination of 3 Values. The values are combined bit-wise.

0 There occurs not output to the log-file.

1 All information will be traced. Information means no errors, nevertheless they documents single fields inside the DLL function.

2 This value defines to trace notes. Notes are useful, if values was changed inside the DLL functions.

4 Use this value to trace errors.

8 Use this value to display additional to the text file message dialogs. The displaying of the message dialogs can be aborted within every message dialog.

Note:

All records will be traced, if you use the value 15 (1 + 2 + 4 + 8).

8.4 Return codes

0	Everything was successful
-1	The parameter ***Inhalt*** is out of the range of 0 to 15.

9 SepaTools_Init

9.1 Purpose of the function

About this feature are made basic initialization of the API. This is the first function which has to be called.

9.2 Function call

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

9.3 Parameters

DataPath Please pass here a pointer to the directory in which the databases are located. These are the files Sepa.dat, Sepa.idx, BLZ.dat and BLZ.idx.

These databases are needed for the proper function of the API. If here is no path passed, the databases will be searched in the current directory.

Please note:

In the case, you do not pass an explicit path, you must not change the working directory during your program flow!

The directory name can have a maximum length of 200 characters.

TempPath Please pass here a pointer to the location in which temporary files are stored in.

If you pass an empty string here, a temporary directory is automatically determined.

The directory name can have a maximum length of 200 characters.

UserPath Please pass here a pointer to a directory in which user data can be stored in (optionally).

User data is used for example if there should be used different BICs and IBAN. User data is used, if bank account number and BLZ is converted to IBAN and BIC and there should be used a different BIC or IBAN.

If you do not want this option, please use the value NIL or NULL. In this case the option will not be initialized.

In the case you enter a valid directory then a plausibility check is carried out with corresponding return codes. The directory name can have a maximum length of 200 characters.

If you pass an empty string here, it is created a directory named SepaUserData within the computer profiles. The directory can be found for example in, C:\Documents and Settings\User\Application Data\SepaUserData.

Lizenz The permanent use of the API requires a licensing.

As part of the licensing you get an up to 10-digit license code. This one has to be (null-terminated) placed.

If you want to use the API for testing purposes as a demo version, please place here an empty string. After this the API can be used to evolutionary cases. You will receive, several times a day, a reference to the demo version in case you call the function.

After placing a correct license code, this will be stopped.

Netz The parameter ***Netz*** has no longer a meaning. The database works until now always in network mode.

9.4 Return codes

0	Everything was successful
1	Everything was successful. However, your databases (Sepa.dat BLZ.dat) are out of date. Please check the database update on the function SepaTools_Info. Current database, see www.sepa-tool.de. **The positive return code is only a note, not an error. Please see also the function SepaTools_Info.**
-1	The directory of the data directory (DataPath) does not exist.
-2	The directory (TempPath) for temporary data does not exist.
-3	The transferred license key is invalid. To test the API, you should pass an empty string.
-4	The database initialization failed.
-5	The path to the data directory (DataPath) is too long, longer than 200 characters.
-6	The path to the temporary directory (TempPath) is too long, longer than 200 characters.
-7	The named drive for the data directory (DataPath) is not ready.
-8	The named drive for the temporary directory (TempPath) is not ready.
-9	The database with the Sepa-data (Sepa.dat) could not be opened. Please check the parameters DataPath.

-10	The database with the stock of BLZ of the Deutsche Bundesbank (BLZ.dat) could not be opened. Please check the parameters DataPath.
-11	The path to the directory for user data (UserPath) is too long, longer than 200 characters.
-12	The named drive for the directory for user data (UserPath) is not ready.
-13	The directory for user data (UserPath) does not exist.
-14	As a directory an empty string was passed. The directory in the profile was not found.
-15	The directory SepaUserData could not be created.
-16	Directory for user data is not writeable.
-17	The database for user data could not be created or opened.
-18	The demo version was only valid for 60 days. This period has expired. Please buy a license to continue.
-19	Databases do not fit to the current program version. Please look for the current DLL and the current databases (www.sepa-tool.de).

10 SepaTools_Free

10.1 Purpose of the function

This function exits the API. All allocated memory areas (by the function SepaTools_Init) are released. It is the last function that was to be called.

10.2 Function call

void SepaTools_Free()

10.3 Parameter

none.

10.4 Return codes

none.

11 SepaTools_SetOptions

11.1 Purpose of the function

With this function, you can set additional global options. If you use this function, you should call them immediately after the call of the function SepaTools_Init.

The call of the function SepaTools_SetOptions is optional.

11.2 Function call

int SepaTools_SetOptions(XMLOptionStruct *OptionStruct)

11.3 Parameters

OptionStruct this is a pointer to the structure XMLOptionStruct. In this structure the parameters are passed.

The structure is shown below.

struct XMLOptionStruct

{	int	FullNameSpace	Value=1, if full Namespace is wanted. [1)]
	char	ConvertFile[255+1]	Path and filename for an additional convert-file. [2)]
	int	ConvertFlag	Different control options [3)]
	Int	UebwSicht	Credit transfer is due immediately [4)]
	int	EilUebw	Same-day credit transfer [5)]
	int	NumMsgID	Numerical Message-Id [6)].
	int	FullPurp	No purpose change [7)]
	int	SammlerTrennen	Create a XML-file for every payment collector [8).]
	int	MaxXMLSatz	Maximal number of XML records in one file [9)]
	int	NotNameOrt	Name and location will not be changed [10)]
	int	CompressXML	Create a XML-file in compact mode [11)]
	int	Umlaute	Value=0, convert umlauts, value=1, pass umlauts
	int	CheckSlash	Value=0, don't check slash, value=1, check slash [12)]
	int	ISOSonder	Value=1 for instant payments according to ISO2017 [13)]
	int	MT940KontoStart	Only used for Camt module.
	int	MT940KontoLen	Only used for Camt module.
	int	MT940KontoMinLen	In effect only if account number longer as this value [14)]
	int	MT940MoreBuTag	More booking Dates in one file [15)]
	int	CamtVersion	Only used for Camt module
	int	NoBlankZweck	No blanks in added parts of the remittance [16)]
	char	Reserve[936]	Reserved for later use.
}			

Additional note:

1) By creating the XML-file, normally a short form of the namespace (without validation information) is used. If you want to use the complete namespace, please pass the value 1.

2) If you pass in this parameter and valid path and a valid filename, an additional convert-file will be created if you convert a bank routing number and a bank account number to BIC and IBAN.

   The file looks like this example.

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

   Affected are the functions SepaTools_WriteXML and SepaTools_WriteDTAtoXML.

   Entries in the file are only written, if actually a conversion was made. If BIC and IBAN are directly passed and used, there are no entries made in this file.

   The meaning of the result codes are explained in the file ZKASpec.pdf, you will find under www.sepa-tool.de under item “Sonstige Infos”.

3) The value of this parameter serves for different control possibilities. It has to be stated bit wise. So it is possible to store more control options in this value.

   At the moment there are the following options (collective use is possible) defined.

   1 At the function SepaTools_ConvertBLZKonto the structure XMLConvertStruct is normally only filled if the call of the function was successful. If there was detected a wrong test digit, the structure stays empty.

     If in the function SepaTools_SetOptions in the field ConvertFlag the value of 1 is delivered, the IBAN also is calculated, even it can’t be used because of errors.

     The origin returncode of the function SepaTools_ConvertBLZKonto is not changed.

   2 Is the value of 2 delivered (as Or-connection), in the functions WriteXML and WriteXMLExt there will be no claim because of faulty BICs and IBANs. They will be written into the XML file anyway. This option acts global until it is changed again.

     If you use the function XMLWriteExt, you are able to state the corresponding value in the structure XMLWriteExtStruc (alternative possibility). With this the control possibility doesn’t act global, but for each data record.

   4 If the value of 4 is delivered (as or-connection), the functions XMLWrite and XMLWriteExt deliver the usual returncodes, but the data record is not written into the XML-file.

     With this option by the functions XMLWrite und XMLWriteExt you are able to proof a data structure, if it is correct without writing it into the XML-file.

     This option acts global, until it is changed again. If you use the function XMLWriteExt you are able to state the corresponding value in the structure XMLWriteExtStruc (alternative possibility). With this the control possibility doesn’t act global, but for each data record.

4) If there are XML-credit transfer data (generated with SEPA Tools), which should be transferred by HBCI(FinTS) to the bank, it is possible that the program which should transfer it, doesn’t accept credit transfers with an execution date. Only credit transfers which are due immediately are accepted.

In this case, please set the parameter “UebwSicht” to the value of 1. The execution date is internally and automatically set to 01.01.1999 (this date is defined for the execution date in such cases). With this, the credit transfer is due immediately (for HBCI).

This concerns the functions SepaTools_WriteXML and SEPATools_WriteDTAtoXML.

5) If you use at least version 2.7 as XML-version, you can achieve that the credit transfer is booked on the same day at the recipient. For this you have to set this variable to the value of 1.

This is no SEPA payment but an express foreign payment. The bank also prices this different. The SEPA XML-format is only used as a means of transport. This is an „Additional Optional Service“ (AOS), which is independent of the SEPA rules.

If you set this field to the value “2”, then Instant payment would be created. For more information, please see http://sepa-tools.de/instant-uebereisung.html.

If you set this variable to 1 or 2, it has consequence for the functions SepaTools_WriteXML, SepaTools_ConvertCSVtoXML and SepaTools_WriteDTAtoXML.

6) The message-ID for an XML-file can be up to 35 characters (alpha numeric). Differing from this, some Austrian electronic banking programs demand for an only numeric message-ID.

If you set these variables to the value of 1, you are able to force this. It is created an up to 20 characters long unique and only numeric message-ID.

This is independent of identifiers, which can be predetermined by the application.

If you set these variables to 1, it has consequence for the functions SepaTools_WriteXML and SepaTools_WriteDTAtoXML.

7) Normally the two lines of purpose will be stripped (leading and trailing blanks). If you don’t like this, then you can set this value to 1.

Although the blank (usual added, if the two lines of purposes are combined), will be repressed.

8) In the normal case, you can create unlimited payment collectors in one XML-file. Some banking applications can only transmit XML-files which contains one payment collector.

In this case, for each payment collector will be created an own XML-file. To do this, set the value to 1. The created filenames are derived from the original passed filename.

9) With this option, you can limit the maximum XML-records in one XML-file. If the limit is reached, than a new XML-file will be created. The passed number must be in the range from 1 to 100.000. Other values will be ignored.

The created filenames are derived from the original passed filename.

10) By using the functions SepaTools_GetBankInfo, SepaTools_GetSEPABank and SepaTools_GetBLZKonto there is a look for the banks dataset by passing a BIC. The search occurs in the SCL-directory of Deutsche Bundesbank.

In the SCL-directory there was sometimes different bank names and bank locations reported, as in the German bank routing stock (BLZ-directory). Because this, there is a check of different bank names or different bank locations between this two directories.

In the normal case the variable NotNameOrt has the value 0 (zero). For this three named functions, you can modify this behavior. The variable can be set to the following values. The values are bit coded (OR connection).

1 For the function SepaTools_GetBankInfo, the in the SCL-directory founded bank name will **not** be overwritten which a different found bank name in the BLZ-directory (bank routing stock).

2 For the function SepaTools_GetBankInfo, the in the SCL-directory founded bank location will **not** be overwritten which a different found bank location in the BLZ-directory (bank routing stock). Exception: In the SCL-directory was no bank location found. In this case the bank location from the BLZ-directory is used.

4 The same method is used for the function SepaTools_GetSEPABank in reference to the bank name.

8 The same method is used for the function SepaTools_GetSEPABank in reference to the bank location.

16 The same method is used for the function SepaTools_GetBLZKonto in reference to the bank name.

32 The same method is used for the function SepaTools_GetBLZKonto in reference to the bank location.

11 Normally the XML-file is created in a readable format. This means that there includes line feeds and spaces for a better reading.

If you set this value to 1, then the xml-file will be created in a compact format (without line feeds and spaces).

12 Since the DK-Version 3.0, for most of the fields are restrictions for the slashes. This is not used for example for the remittance, because primary for references.

These fields must not end or begin with a slash. Even it is not allowed, that it contains double slashes, but only single slashes.

With this option, you can define that not allowed slashes (since DK-Version 3.0) will be removed.

The check of the allowed chars is on many places in SepaTools implemented. So this option works for all data fields. If you will difference between remittance and references, you should not set this option and manage this point by your self.

13 By setting of this field to the value “1”, you can create an instant payment according to the scheme pain.001.001.008.xsd. Additional to the execution date, there can be passed an execution time.

More information about instant payments, are available at the page http://sepa-tools.de/instant-ueberweisungen.html

14 See German documentation

15 See German documentation

16 No blanks between the parts of the remittance.

Returncodes

0	Everything was successful
-1	The passed path (incl. filename) was too long (>200 characters).
-2	The specified drive in the path was not ready.
-3	The directory does not exist.
-4	The handed path is write-protected.
-999	The API was not initializes. Please call first the function “SepaTools_Init“.

12 SepaTools_Info

12.1 Purpose of the function

This function returns information about the actuality of the database update Sepa.dat and BLZ.dat.

12.2 Function call

int SepaTools_Info(XMLInfoStruct *InfoStruct)

12.3 Parameters

InfoStruct This is a pointer to the structure XMLInfoStruct. In this structure, the information is returned.

The structure is shown below.

```
struct   XMLInfoStruct

{   char    EBADatum[8+1]      Date of the actuality of the available banks (DDMMYYYY).
    char    BLZVon[8+1]        BLZ-stock, valid from this date. Format DDMMYYYY.
    char    BLZBis[8+1]        BLZ-stock, valid until this date. Format DDMMYYYY.
}
```

12.4 Return codes

0	Everything was successful.
-1	The information could not be read from the database. Please check the data path which you have placed with the Sepa-Tools_Init

-999	The API has not been initialized. Please first call the function SepaTools_Init.

13 Create XML files

The API can create SEPA XML files. Three functions are necessary for this. The logic is running as follows represents:

- function SepaTools_CreateXML	Starts and initializes the process.
- function SepaTools_WriteXML	Repeat this function call for every record with payment data. The limit for the number of calls is only the size of your memory area.
- function SepaTools_CloseXML	The process will be closed. The file is written to the specified data carrier.

Within this function will be held extensive plausibility checks.

14 SepaTools_CreateXML

14.1 Purpose of the function

This function initializes then XML export. The corresponding parameters in the structure are passed.

14.2 Function call

int SepaTools_CreateXML(XMLCreateStruct *CreateStruct)

14.3 Parameter

CreateStruct Please pass a pointer to CreateStruct. The structure is shown below. All strings are terminated with a binary NULL. The length of the char array is therefore one byte longer as the actual text.

For all other strings it is checked internally if it is a valid character set for SEPA payments. Is it not, the invalid characters are removed. The corrected character string is returned in the structure.

Already this function checks whether in the specified path for the export file (the file which has to be emitted), can be written.

```
struct    XMLCreateStruct

{   char    ExportPfad[255+1]       Drive and path for export (XML-file).
    int     Direct debit            Credit transfer(=0) oder debit direct (=1).
    char    XMLName[35+1]           The name of the XML-file to create 1).
    char    MsgId[35+1]             Message-Id for the XML-file 2).
    char    EinreicherName[70+1]    Name of the presenter of the file (at least 3 chars).
}
```

Additional note:

1) Regardless of the passed file name extension or not passed file name extension the file name extension is always set to .xml.

   If an empty string (or the value Dummy.txt) is passed, the filename is managed internally by the API. This is necessary if the XML data should be exported to memory as a byte-array.

2) The Message-Id should be a clear identification of the possible XML file. If you pass an empty string here, it is automatically internally formed a Message-Id, and returned in the structure.

14.4 Return codes

0	Everything was successful
-1	The path for the export file is to short (< 2 chars).
-2	The specified directory for the export file does not exist.
-3	The data carrier for the export file is write-protected. The file can't be written.
-4	The name of the presenter is to short (less than 3 chars).
-5	For the process required temporary file could not be opened. Please check the parameter in the function "SepaTools_Init".
-6	The pathname for the export file is too long (>200 chars).
-7	The specified drive for the export file is not ready. (possibly no data carrier is inserted).
-8	The name for the XML-file is too short. (<3 chars).
-9	The function „SepaTools_CreateXML" was already called. Before a second call, the function „SepaTools_CloseXML" must be called. The functions to create XML-files are not „Multi-Thread" capable!
-999	The API was not initializes. Please call first the function "SepaTools_Init".

15 SepaTools_CreateXMLExt

15.1 Purpose of the function

This function initializes then XML export. The corresponding parameters in the structure are passed.

In opposite to the function SepaTools_CreateXML, this function provides a block of reserved chars for later use. So we can expand the function without the necessity to change the structure.

We recommend you to use this function for future use.

15.2 Function call

int SepaTools_CreateXMLExt(XMLCreateExtStruct *CreateStruct)

15.3 Parameter

CreateStruct Please pass a pointer to CreateStruct. The structure is shown below. All strings are terminated with a binary NULL. The length of the char array is therefore one byte longer as the actual text.

For all other strings it is checked internally if it is a valid character set for SEPA payments. Is it not, the invalid characters are removed. The corrected character string is returned in the structure.

Already this function checks whether in the specified path for the export file (the file which has to be emitted), can be written.

struct XMLCreateExtStruct

{	char	ExportPfad[255+1]	Drive and path for export (XML-file).
	int	Direct debit	Credit transfer(=0) oder debit direct (=1).
	char	XMLName[35+1]	The name of the XML-file to create [1)].
	char	MsgId[35+1]	Message-Id for the XML-file [2)].
	char	EinreicherName[70+1]	Name of the presenter of the file (at least 3 chars).
	int	DoWhgSort	Value 1 for additional currency sort [3)]
	int	PmtTypeInfoInTx	Value 1 for PmtTpInf inside the transaction level [4)]
	char	CtctDtls_Nm_CH[35+1]	Contact details for GroupHeader, only Switzerland [5)]
	char	CtctDtls_Other_CH[35+1]	Contact details Other Id for GroupHeader, only CH [5)]
	int	ShortMsgId	Numeric 16 characters long value for an Id [6)]
	char	Reserve[1000]	Reserved for later use.
}			

Additional note:

1) Regardless of the passed file name extension or not passed file name extension the file name extension is always set to .xml.

   If an empty string (or the value Dummy.txt) is passed, the filename is managed internally by the API. This is necessary if the XML data should be exported to memory as a byte-array.

2) The Message-Id should be a clear identification of the possible XML file. If you pass an empty string here, it is automatically internally formed a Message-Id, and returned in the structure.

3) Normally, a sort of the records according the currency is not necessary, because the currency is always EUR. In Switzerland, there can be different currencies. If you set this vale to 1 (default=0), then there would be an additional sort for the currency. This means, by changing the currency a grouping occurs (on Payment Information Block for each currency).

4) Generally the PmtTpInfo Block is shown in the Payment Information Block. In the PmtTpInfo is also the value for the sequence of Direct Debit payments (i.e. FRST or RCUR). This means, that by changing the sequence, it would be created a new Payment Information Block. The Creditor gets one booking for each Payment Information Block on his account.

   If you set this value to 1, then the PmtTpInfo will be placed in the Transaction-area. Now you can mix the sequences and only one Payment Information Block will be created.

   This only works on DK-version 3.1 and up.

5) In Switzerland it is possible to place additional contact details in the Group Header. The field CtctDtls_Nm_CH must not be empty, it the values should be written. The field CtctDtls_Other_CH is only written, if both fields are not empty.

6) The PaymentInfoId and the EndToEndId are defined as a string witch up to 35 characters. In practice we see, that some banks have an internal restriction. They only accept strings with a length of maximal 16 characters.

   In behave to this restriction you can assign a bit coded value to this field (or connection).

   1 The EndToEndId will be created to a 16 characters long number.

   2 The PmaymentInfoId will be created to a 16 characters long number.

   If you like to have both Ids with 16 characters long numbers, then you have to pass the value 3.

15.4 Return codes

0	Everything was successful
-1	The path for the export file is to short (< 2 chars).
-2	The specified directory for the export file does not exist.
-3	The data carrier for the export file is write-protected. The file can’t be written.
-4	The name of the presenter is to short (less than 3 chars).
-5	For the process required temporary file could not be opened. Please check the parameter in the function “SepaTools_Init”.

-6	The pathname for the export file is too long (>200 chars).
-7	The specified drive for the export file is not ready. (possibily no data carrier is inserted).
-8	The name for the XML-file is too short. (<3 chars).
-9	The function „SepaTools_CreateXML“ was already called. Before a second call, the function „SepaTools_CloseXML” must be called. The functions to create XML-files are not „Multi-Thread“ capable!
-999	The API was not initializes. Please call first the function “SepaTools_Init“.

16 SepaTools_WriteXML

16.1 Purpose of the function

On each call to this function exactly one record for a payment order is passed. There are extensive plausibility checks.

Are no BIC and no IBAN delivered or both values are invalid, so a German bank routing number and a German account number can be passed as replacement.

The measured values are then returned in the structure „WriteStruct".

Have specific BICs and IBANs with the corresponding bank codes and account numbers deposited (e.g. function AddUserBICor AddUserIBAN) with, so this translation table is included in the determination of BIC and IBAN for bank routing number and account number.

Caution:

In this automatic calculation errors can occur. A liability is not accepted. It is recommended to deliver BIC and IBAN always directly and correctly.

It can be mixed payment orders are passed by different customers. The orders are automatically sorted by customers, so that multiple payment transactions from the same customer, the orders internally automatically are sorted and collected.

16.2 Function call

int SepaTools_WriteXML(XMLWriteStruct *WriteStruct)

16.3 Parameter

WriteStruct Pass here a pointer to the structure „WriteStruct". The structure is shown below. All strings are passed null-terminated. The length of the char-array is always 1 byte longer as the actual text.

For all strings is checked internally whether it is a valid character set for SEPA payments. This is not the case, the invalid characters are removed. The corrected string is then returned in the structure

Banking expertise

The initiator of a payment is for credit transfer the payer of the order. For direct debit, the initiator is the recipient of the payment.

For credit transfer the recipient of the payment is the part get the payment. For direct debit, the recipient of the payment is who has to pay.

```
struct    XMLWriteStruct

{   char    PmInfoId[10+1]      PaymentInfoId – Designation of the Order [1)].
    char    AusfDatum[8+1]      Execution date for this payment in the form DDMMYYYY.
    char    AuftragName[70+1]   Name of the initiator, at least 3 chars.
    char    AuftragBIC[11+1]    BIC of the initiator.
```

char	AuftragIBAN[35+1]	IBAN of the initiator.
char	AuftragAbwName[70+1]	Optionally a different name of the initiator.
char	AuftragCI[35+1]	CI of the initiator, only direct debit.
char	AuftragBLZ[8+1]	Optionally (bank routing number) BLZ of the initiator.
char	Auftragkonto[11+1]	Optionally account number of the initiator.
char	EmpfName[70+1]	Name of the recipient, at least 3 chars.
char	EmpfBIC[11+1]	BIC (bank routing number) of the recipient [6)].
char	EmpfIBAN[35+1]	IBAN (account number) of the recipient.
char	EmpfAbwName[70+1]	Optionally a different name of the recipient.
char	EmpfBLZ[8+1]	Optionally the BLZ (bank routing number) of the recipient.
char	EmpfKonto[11+1]	Optionally the account number of the recipient.
char	Purpose[4+1]	Optionally the purpose of the transfer [2)].
char	Betrag[12+1]	Amount of the Transfer as a string.
char	EndToEndId[10+1]	ID of the single transfer [3)].
char	MandatId[35+1]	ID of the mandate, only for direct debit.
char	MandatDatum[8+1]	Date of the mandate in the form DDMMYYYY.
char	SequenceType[4+1]	Sequence, only for direct debit [4)].
int	B2B	Kind of direct debit [5)].
char	Zweck1[70+1]	Purpose line 1
char	Zweck2[70+1]	Purpose line 2

}

Additional notes:

1) To uniquely identify the initiator an internal ID is required. Please pass an up to 10-digit abbreviation for this ID. Internally, the abbreviation is extended with the date, time and a serial number.

   If you pass an empty string here, the complete identification will be automatically formed.

   As a result of a possible internal sorting the PmInfoId is formed at a later point in time in the context of calling “SepaTools_CloseXML”, it could not be returned at this point in the structure. The actual results can be seen only in the generated XML file.

2) The possible values here are shown in the Sepa rulebook. You can leave this field blank (empty string), because it is optional.

3) To uniquely identify the payment order an internal ID is required. Please enter an up to 10-digit abbreviation for this ID. Internally, the abbreviation is then extended with the date, time and a serial number.

   If you pass an empty string here, the complete identification is created automatically.

   As a result of the possible internal sorting EndToEndId is formed only at a later time in the context of the call to “SepaTools_CloseXML”, they it could not returned at this point to the in the structure. The actual result can be seen in the XML file.

4) Here you can specify the frequency with which direct debits are to be executed. Valid values are here:

OOFF	Singular direct debit.
FRST	First direct debit.
RCUR	Periodic submission of direct debit.
FNAL	Last submission of direct debit.

If no value is passed, the value is internally automatically set to RCUR.

5) To regulate the kind of direct debits, you have to set the variables „B2B“.

The following valued are possible.

0 Normal CORE-direct debit with term of presenting 3 or 6 days.
1 Business-to-Business direct debit B2B. This accords to the previously pre-authorized direct deb its.
2 Direct debit (COR1) with shortened term of presenting of 2 days. The shortened term of presenting is valid for both, for first and recurrent direct debits.

You only are allowed to set the value of 2, if the version is set to 2.7 at least, by the function SepaTools_SetVersionUndLand. If this minimum version wasn't set, the value of 2 (if delivered) is internally set to the value of 0.

The type of direct debit COR1 has to be arranged between the credit institutions bilateral. In Germany an Austria, this is arranged.

6) If you use XML-version 2.7 the BIC doesn't have to be delivered. The payment is executed in the mode of „IBAN Only“. This is valid from November 2013 for payments within Germany and it is planed for the whole SEPA area for 2016.

Independence of this you can deliver the BIC:

Internally, it is proofed (provided that you use version 2.7), if the IBAN is valid. If it is valid, the data record is built even without a BIC.

16.4 Return codes

Note:

In the case of return code> = 0 (no error), in the return value, additional information are stored. This information is stored bitwise.

The bit assignment for the value >= 0 in the return value is as follows:

0	Everything was successful.
1	The calculated IBAN for the initiator can be used. But it is not unique. An inquiry with the customer is advised. This positive return code can currently only occur in 7-digit accounts of Deutsche Bank.
2	The IBAN can be calculated for the recipient can be used. But it is not unique. An inquiry with the recipient is advised.

From here, the error codes are evaluated as negative integers again.

-1	The writing in the temporary file failed.

-2	The function „SepaTools_CreateXML“ was not yet called.
-101	The name of the recipient was not passed, because have less than 3 characters.
-102	The BIC (Bank Identifier Code) for then recipient was formally invalid.
-103	No IBAN was passed or the IBAN is wrong. The IBAN is not usable.
-104	No amount was passed or the amount is less 0 (negative).
-106	The bank with the passed BIC-code of the payee takes not part on SEPA-transfer scheme.
-107	The bank with the passed BIC-code of the payee takes not part on SEPA-direct debit scheme.
-108	The country ID (part of IBAN), which was passed for the payee is not correct for a SEPA-country. The IBAN is not usable.
-109	The BIC-code of the recipient is not listed in the list of the reachable banks. The BIC-code is not usable.
-110	For the „BLZ“ (bank routing number) for the recipient, IBAN could not be determined.
-111	The determination of the IBAN from bank routing number and account number of the recipient is not unique. The IBAN could not be determined.
-112	The check digit from the passed account number is wrong. A determining of the IBAN is not possible.
-113	The delivered bank routing number for the recipient is marked for deletion and could not be used for determining of BIC and IBAN.
-114	The bank with the passed BIC-code (recipient) takes not part of the SEPA-B2B direct debit scheme.
-120	There was only one direct debit with the type CORE created. Different types of the direct debit in one file are not allowed.
-121	There was only one direct debit with the type B2B created. Different types of the direct debit in one file are not allowed.
-122	There was only one direct debit with the type COR1 created. Different types of the direct debit in one file are not allowed.
-123	The bank with the passed BIC-code (recipient) takes not part of the SEPA-COR1 direct debit scheme.
-151	The delivered name for the recipient has less than 3 chars.

-152	For the initiator, either no BIC code was passed, or the BIC code is wrong. This error is also generated when you want to convert BLZ and account number in IBAN and BIC (no BIC and no IBAN is passed) and the BLZ is invalid. In this case no BIC can be determined.
-153	There was ether no IBAN delivered or the IBAN is incorrect. The IBAN can't be used.
-156	The bank of the given BIC code of the initiator doesn't participate in the SEPA Credit Transfer Scheme.
-157	The bank of the given BIC code of the client doesn't participate in the SEPA Direct Debit.
-158	The country code, which was passed in the IBAN for the client is not a SEPA country. The IBAN can't be used.
-159	You want to create direct debits. Ether there was no CI passed or the CI isn't valid.
-160	Ether you have passed no mandate for direct debits or the mandate consists of invalid characters.
-161	Ether you have passed no date of the mandate for direct debits or the date of the mandate is invalid.
-163	The execution date is invalid, or it is past.
-164	The BIC code of the initiator is not listed in the list of reachable banks. The BIC can't be used.
-165	For the BIC which was passed by the initiator couldn't be determined an IBAN.
-166	Determination of the IBAN with BLZ and account number is not unique. The IBAN can't be determined.
-167	The initiators account number has a wrong check digit. A determination of the IBAN is not possible.
-168	The delivered BLZ will be deleted (flagged). It can't be used to determine BIC and IBAN.
-169	The transferred sequence type is invalid. In the case of doubt please pass here an empty string.
-170	The Bank of the given BIC code of the client doesn't participate on the SEPA B2B Direct Debit Scheme.
-999	The API has not been initialized. First please call the function SepaTools_Init.

17 SepaTools_WriteXMLExt

17.1 Purpose of the function

This is an extended function of the function SepaTools_WriteXML. With this a conceptual error in the function SepaTools_WriteXML is corrected and additional data fields are fit in.

The function SepaTools_WriteXML will be unchanged and without change of the data structure conserved. At the extended data structure XMLWriteExtStruct only the additional data fields are described. Data fields, functionalities of the function, and returncodes which are not described are identic with the origin function SepaTools_WriteXML.

17.2 Function call

int SepaTools_WriteXMLExt(XMLWriteExtStruct *WriteStruct)

17.3 Parameter

WriteStruct Here a pointer is delivered to the structure XMLWriteExtStruct. The structure is shown in the following. All the strings are delivered null terminated. The length of the character arrays is always one character longer than the real array.

There is proofed internally (concerning all character arrays), if it is a valid character set for SEPA payments. If it is not valid, the invalid characters are deleted. The corrected characters array is delivered in the structure.

Only the additional data fields are documented.

Because of the support to create xml-files in Switzerland, the data structure was modified and extended. Because of the differentiated control of the payment in Switzerland, this support needs more data fields in the data structure.

If you don’t use the fields ***StructZweck***, ***StructTyp*** and ***BatchBooking***, you have nothing to do or to adapt.

The additional fields are placed in the reserve area. The complete size of the structure is not changed.

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	Value=1, if no check of BIC and IBAN [1)]
int	OnlyCheck	Value=1, if the data set should not be written [2)]
char	OrgName[70+1]	origin name of the presenter of the direct debit
char	OrgMandat[35+1]	origin mandate-Id
char	OrgCI[35+1]	origin CI of the presenter of the direct debit
char	OrgIBAN[35+1]	origin IBAN of the payer
int	UseSMNDA	Value=1, if SMNDA use note [3)]
int	DoStruct	Value=1, for structured purpose [4)]
char	StructZweck[35+1]	Structured purpose [4])]
char	StructTyp[4+1]	Type of the structured remittance Information [6)]
int	BatchBooking	To set an explicit BatchBooking [5)]
char	InstrId[35+1]	Unique transaction reference [7)]
char	Whg[3+1]	Optional Currency [8)]
char	OrgBIC[11+1]	origin BIC of the payer
char	EmpfLand[2+1]	Optional the country code of the recipient [23)]
char	EmpfAdrLine1_Str[35+1]	Option., address line 1 of the recipient [23)]
char	EmpfAdrLine2_Ort[35+1]	Option., address line 2 of the recipient [23)]
char	AuftragLand[2+1]	Optional the country code of the initiator [24)]
char	AuftragAdrLine1_Str[35+1]	Option., address line 1 of the initiator [24)]
char	AuftragAdrLine2_Ort[35+1]	Option., address line 2 of the initiator [24)]
char	CategoryPurpose[4+1]	Optional category purpose code, 4 character
char	AusfZeit[6+1]	Optional execution time for instant payments [29)]
char	StructPrtry[35+1]	Optional priority for structured rem. Information
char	EmpfAdrHausnummer[10+1]	Optional the building number of the recipient [23)]
char	EmpfAdrPLZ[15+1]	Optional the ZIP code of the recipient [23)]
char	AuftragAdrHausnummer[10+1]	Optional the building number of the initiator [24)]
char	AuftragAdrPLZ[15+1]	Optional the ZIP code of the initiator [24)]
char	Reserve1[292]	Reserved for later use

Int	EmpfAdrStruct	Use structured address [23)]
int	AuftragAdrStruct	Use structured address [24)]
int	ChBankAdrStruct	Use structured Address for bank [19)]
int	ReserveInt[12]	Reserved for later use (Field with 12 Integer)
int	CHZVArt	Kind of Payment for presenting in Switzerland [9)]
char	ChKonto[15+1]	Bank account number or postal account number [10)]
char	ChBankBC[10+1]	Bank clearing code (creditor/debtor) CH [11)]
char	ChBankPost[15+1]	Postal account creditor/debtor (Ch) [12)]
char	ChBankName[35+1]	Name of the bank creditor/debtor) (Ch) [13)]
char	ChBankLand[2+1]	Country of the bank creditor/debtor (Ch) [14)]
char	ChBankAdrLine1_Str[35+1]	Address line 1 of the bank creditor/debtor (Ch) [15)]
char	ChBankAdrLine2_Ort[35+1]	City of the Bank creditor/debtor (Ch) [16)]
char	ChBankAdrPLZ[15+1]	ZIP code of the address of the bank (Ch) [17)]
char	ChBankHausnummer[10+1]	Building number of the bank (Ch) [18)]
char	ChReserve[57]	Reserved for later use.
char	ChBankClearingId[5+1]	Clearing Id for foreign payments (Ch) [20)]
char	ChBankIniId[35+1]	Id of the presenter (Ch) [21)]
char	ChLSCreditorBC[10+1]	Bank clearing code of the presenter (CH) [22)]
char	ChAccountPrtry[35+1]	Additional Information for the initiator (CH) [25)]
char	ChLSCreditorESR[35+1]	ESR-Number of the Creditor (CH) [26)]
int	ChPmtTpInf	PaymentTpInfo in Level B (CH) [27)]
int	ChNoZVArtSort	No Grouping to Swiss payment kind [28)]
char	CHQRInfo[35+1]	Additional Information for QR Invoices
char	Reserve2[384]	Reserved for later use.

}

Additional notes:

1) If you set this value (on individual record) to 1, the payments are also written into the XML-file even BIC or IBAN of the recipient or initiator are wrong.

   Alternative you can achieve this by a global Option with the call of the function SetOptions.

2) If you set this (on individual record) to the value 1, this data record will not be written into the XML file. the formal checks will be done. Because of this the returncodes are the same.

   Alternative you can achieve this by a global Option in the call of the function SetOptions.

   Note:

   The following information should inform the payer about a change concerning the mandate (e.g.: another CI or a new mandate-Id). In which extent and in which manner those information from the bank of the payer are given to the payer (with the statements of account) corresponds to the bank.

3) If this value is set to 1, in the XML file it is indexed with the value SMNDA (**S**ame **M**andat with **N**ew **D**ebtor **A**gent) that an origin mandate is used with a new bank. This value only may be used with the sequence of direct debits FIRST.

   At the end of 2016 this rule has been changed with DK-version 3.0. This is managed in the DLL automatically.

   The experience shows, that not all countries has realized this change. To force the former behavior, please pass the value 2 to this field.

4) Instead of the unstructured purpose, you can use the structured purpose. In this case, you have to set the variable DoStruct to the value 1. To the variable StructZweck, you can pass the structured purpose.

   Caution:

   There is no validation check of the structured purpose. You can only use the structured **or** then unstructured purpose, **not both**.

5) In the normal case, there is not setting for the value BatchBooking in the payment-information. The booking of the payments to the account of the presenter, is depended by the pre settings of the bank of the presenter.

   With the value BatchBooking, you can explicit set the behavior. The following values are valid:

   1 BatchBooking will be set to the value false. This means, every item of the bookings will booked as a single.

   2 BatchBoking will set to the value true. This means, there is only one booking.

   Attention please:

   The effect of this value is depended of agreements with the bank of the presenter.

6) If no value is passed in this field, automatically the value SCOR is set internally. In Germany and Austria this is the only valid value.

   Exception:

   By using the Switzerland payment kind 1 (ESR-Payment), this field mat be empty (no value).

   By using direct debit in Switzerland (payment kind 102), the value IPI or ESR is to pass here.

7) In regard to the guide line, this value should only be passed by a technical service provider. It should be an own reference. In Switzerland it is recommended to fill this field with a value.

8) This field should only be used, if the XML-file is presented in Switzerland (see CH documentation). If this field is not filled, then the value EUR will be set automatically.

9) This field is only valid for presenting in Switzerland. The following values are allowed.

   0 This is standard, if not Switzerland.
   1 Payment kind 1, regards to Switzerland documentation.
   21 Payment kind 2.1, regards to Switzerland documentation.
   22 Payment kind 2.2, regards to Switzerland documentation.
   3 Payment kind 3, regards to Switzerland documentation.
   4 Payment kind 4, regards to Switzerland documentation.
   5 Payment kind 5, regards to Switzerland documentation.
   6 Payment kind 6, regards to Switzerland documentation.
   101 Direct debit for postal finance in Switzerland (DD).
   102 Direct debit for the other banks in Switzerland (TA).
   103 Direct debit for postal finance in Switzerland (DD).

10) Please pass here for payment orders the bank account number or the postal finance account number, if you don't like or could not use an IBAN.

11) For addressing the creditor or the debtor in Switzerland will frequently passed the bank clearing code.

12) For addressing the creditor or the debtor in Switzerland will frequently passed the postal bank account number.

13) Depending from the kind of payment, the name of the bank of the creditor/debtor must be passed in Switzerland.

14) Depending from the kind of payment, the country of the bank of the creditor/debtor must be passed in Switzerland. If this field by presenting in Switzerland not filled, the value CH will be set internally automatically.

15) Depending from the kind of payment, the address of the bank of the creditor/debtor must be passed in Switzerland. If the address should be passed in structured form (in Switzerland recommended), so the City of the bank of the creditor/debtor is to pass here.

    If you don't a value to the field ***ChBankAdrStruct***, this value is used as address line 1.

16) Depending from the kind of payment, the address of the bank of the creditor/debtor must be passed in Switzerland. If the address should be passed in structured form (in Switzerland recommended), so the name of the city of the bank of the creditor/debtor is to pass here.

    If you don't a value to the field ***ChBankAdrStruct***, this value is used as address line 2.

17) Depending from the kind of payment, the address of the bank of the creditor/debtor must be passed in Switzerland. If the address should be passed in structured form (in Switzerland recommended), so the ZIP code of the bank of the creditor/debtor is to pass here.

18) Depending from the kind of payment, the address of the bank of the creditor/debtor must be passed in Switzerland. If the address should be passed in structured form (in Switzerland recommended), so the building number of the bank of the creditor/debtor is to pass here.

19) Please pass the value **1** to this field, if you want to use the structured address. If you don't pass a value to this field (or the value **0**), then the unstructured address is used.

20) By using the payment kind of 6 (foreign payment, not SEPA), the clearing-id regarding to the External code sets is to use. For example, for Germany this is DEBLZ.

21) Inside the Group-Header in the XML-file can or must be passed an Id for the presenter of the payment. Using transfer credit, this is optional. Using direct debit (payment kind 101 and 102), it is forced necessary. This Id is to coordinate with the bank of the presenter of the direct debit payment.

22) By using the payment kind 102 (direct debit other banks), the creditor will not be identified with a BIC, else with a bank clearing code. This code is to pass here.

23) You can pass optionally up to two address lines for the recipient. If you do this, the passing of the country code is mandatory.

You also can pass only the country code.

To use the unstructured address, please fill optional the fields ***EmpfAdrLine1_Str*** and ***EmpfAdrLine2_Ort***.

If you would like to use the structured address, the field ***EmpfAdrLine1_Str*** represents the street of the address. The field ***EmpfAdrLine2_Ort*** represents the City of the address.

Additional you can fill the field ***EmpfAdrHausnummer*** (Building number) and the field ***EmpfAdrPLZ*** (ZIP code).

To show the structured address in the XML file, it is necessary to set the field ***EmpfAdrStruct*** to the value of 1. Otherwise the unstructured address is used.

Please note:

Different to the ISO norm, in Germany the structured address is not allowed. The structured address will raise an error.

By presenting the XML file in other countries the structured address is normally allowed. In Switzerland it is necessary since the year 2025.

24) You can pass optionally up to two address lines for the client. If you do this, the passing of the country code is mandatory.

You also can pass only the country code.

To use the unstructured address, please fill optional the fields ***AuftragAdrLine1_Str*** and ***AuftragAdrLine2_Ort***.

If you would like to use the structured address, the field ***AuftragAdrLine1_Str*** represents the street of the address. The field ***AuftragAdrLine2_Ort*** represents the City of the address.

Additional you can fill the field ***AuftragAdrHausnummer*** (Building number) and the field ***AuftragAdrPLZ*** (ZIP code).

To show the structured address in the XML file, it is necessary to set the field ***AuftragAdrStruct*** to the value of 1. Otherwise the unstructured address is used.

Please note:

Different to the ISO norm, in Germany the structured address is not allowed. The structured address will raise an error.

By presenting the XML file in other countries the structured address is normally allowed. In Switzerland it is necessary since the year 2025.

25) For Switzerland it is possible to pass additional information for the account of the initiator (i.e. for the kind of the booking, i.e. the value CND for the Berner Kantonalbank) to the structure.

If in different payments are values, which are in the first 10 characters different, a new payment information block will be build.

26) If the payment kind 102 (TA Direct Debit) is used and the ESR reference is given, then in this field must be placed the ESR-number of the submitter.

27) Payment-Type-Information can be written in level B or level C of the XML file. The recommend of SIX is, it to write in level C for the payment kinds 1, 2.1 and 2.2.

    Some banks have problems with this behavior. If you pass the value “1” to this field, this information will be placed in level B.

28) If you pass the value “1” to this field, no grouping to Swiss payment kinds will happen.

29) With using the Option ***ISOSonder*** within the function ***SetOptions*** you can pass a time stamp for the execution of the Instant payment. It will create a version pain.001.001.08.

    You can also use this field with the same meaning with the version DK 3.7. This is in SepaTools the version 8.

    The Option ***ISOSonder*** must **not** be set. A version pain.001.001.09 will be created.

17.4 Returncodes (additionally)

-116	The IBAN which was delivered for the note of the change of the mandate is invalid.
-117	The note of the identifier SMNDA may only be used in connection with the type of sequence FRST.
-118	The origin name of the creditor may not be identic with the current name of the creditor.
-119	The origin mandate may not be identic with the current mandate.
-173	The CI which was delivered to note the change of the mandate, is invalid.
-174	The origin CI may not be identic with the current used CI.
-301	The kind of payment, which is passed, is not valid. For credit transfer, the value must be 1, 21, 22, 3, 4, 5 or 6. For direct debit the value must be 101,102 or 103.
-302	If you use the payment kinds 1 or 102, then you must also use the structured remittance Information.

18 SepaTools_CloseXML

18.1 Purpose of the function

With this function, the actual XML file is generated from the temporary file produced by SepaTools_WriteXML and written to the specified file.

18.2 Function Call

int SepaTools_CloseXML(XMLCloseStruct *CloseStruct)

18.3 Parameters

CloseStruct Here a pointer is passed to the structure CloseStruct. The structure is shown below. The structure is passed empty and filled with information for the application by the API.

```
struct    XMLCloseStruct

{   int      Anzahl                Number of generated data records.
    char     SummeBetrag[12+1]     Total sum of amounts in the XML-file as a character
                                   string in cents.
}
```

18.4 Return codes

0	Everything was successful.
-1	The XML file could not be created/initialized. Please contact the manufacturer.
-2	When creating each record, there appeared an error. Please contact the manufacturer.
-3	When writing the XML file to disk an error has appeared.

The following return codes from -4 to -7 can only occur when operating in "memory mode" (no file name for the XML file has been passed). The XML file is not passed as a file. The content can be retrieved from the memory by the function SepaTools_XMLGetData (see page 68).

-4	There are no data available to output the XML file in the memory.
-5	The size of the XML data is greater than the maximum size allowed (300 MB).
-6	The memory area to transfer the data is invalid. Maybe there is not enough memory available.
-7	The output of the XML data in the memory area failed.
-8	No data records could be written, because all records had errors.
-9	The temporary file could not be opened.

-999	The API has not been initialized. Please first call to the function SepaTools_Init.

19 XML-support for Switzerland

Switzerland has a special stat inside of the SEPA-countries, because Switzerland is neither in the European Union and has no Euro. This has an effect of the payment presenting to banks in Switzerland.

XML-files, which are created for Switzerland, can also be presented to banks in Liechtenstein.

The extensions reflect actual to create XML-files with the function SepaTools_WriteXMLExt.

19.1 Credit Transfer

The XML-format in Switzerland is not restricted to Euro-payment as in Germany and Austria. There a different kinds of payments, which are in part founded in the history of Switzerland payments.

The usual presentation of the payment partners by using of BIC and IBAN is in Switzerland only one possibility of more. As well for the currency can used EUR and CHF.

To create valid XML-files, it is necessary to pass the required kind of payment to the API. These differences are determined in the “Schweizer Implementation Guidelines” or customer to bank messages. This document is placed with kindly approval of the “SIX Interbank Clearing AG” in Zürich at the website www.sepa-tools.de.

Inside one XML-file the different kinds of payment can be mixed. The API performs automatically a sort, in which the kind of payment is included too.

After the sort, there will be created corresponding payment information blocks, so that the same kinds of payments are arranged in on block.

The payment-information-id must be different from one block to another block. The API does this automatically in default. If you use the function SepaTools_SetId, you have to be care for this theme.

There are the following different kinds of payment:

19.1.1 Domestic payments

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 V2: BC und Postkonto der Bank V3: Postkonto der Bank und Name der Bank	V1: BC V2: BIC Inland	V1: BIC Inland V2: BC und Name und Adresse FI V3: Name und Adresse Finanz-institut Inland
Currency	CHF/EUR	CHF/EUR	CHF/EUR	CHF/EUR	Alle ausser CHF/EUR*

1 The SER-method is a deposit method, which is used from the Switzerland postal for companies with are located in Switzerland. A participant of the ESR method gets a ESR part number, which is used in the XML-file instead of the IBAN.

The ESR 27 char length reference number is to pass to the structured remittance Information.

The currency can be EUR or CHF.

2.1 In this case an invoice will be paid with a red payment slip (ES) in favor to postal account, which was sent from the payment receiver to the debtor. Instead of the IBAN, in the XML-file is posted the postal account number of the payment receiver (creditor).

Usually the unstructured remittance Information is to pass.

The currency can be EUR or CHF.

2.2 In this case an invoice will be paid with a red payment slip (ES) in favor to bank account, which was sent from the payment receiver to the debtor.

The account of the payment receiver (creditor) will be addressed with an IBAN (domestic) or a bank account number. To address the account of the creditor, there are three variants which can used optional.

V1 Please pass the bank clearing code.
V2 Please pass the bank clearing code and the postal account number of the payment receiver (creditor).
V3 Please pass then postal account number of the payment receiver and provide the name of the bank of the receiver.

The remittance Information is usually to pass in unstructured form.

The currency can be EUR or CHF.

3 In this case it is a domestic payment in the currency EUR or CHF. For the identification of the payment receiver (creditor), can be used the IBAN, the postal account or the bank account number.

To address the bank of the payment receiver (creditor) you can use optional three variants:

V1 Please provide the bank clearing code.
V2 Please provide the domestic BIC of the bank of the payment receiver (creditor).

The remittance Information is usually to pass in unstructured form.

If you pass BIC and IBAN for the payment receiver combined with payment kind 3 (V2), there is a check if BIC and IBAN belongs to Switzerland or Liechtenstein. In this case the kind of payment will be automatically intern set to the value 5.

4 This kind of payment is used for domestic credit transfer in a foreign currency (all currencies without CHF and EUR).

To identify the payment receiver, the IBAN, the postal account or the bank account can be used.

To address the bank of the payment receiver, there can be used three options:

V1 Providing the domestic BIC of the bank of the payment receiver.
V2 Providing the bank clearing code and the name and the address of the bank for the payment receiver (creditor).
V3 Providing the name and the address of the bank of the payment receiver.

The remittance Information is usually to pass in unstructured form.

19.1.2 Foreign Payments

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 V2: Bankcode (ohne BC) und Name und Adresse Finanzinstitut V3: Name und Adresse Finanzinstitut International
Currency	EUR	alle*

5 In this case, it is a typical European SEPA payment. The currency must be EUR.

The payment receiver is to address by using BIC and IBAN.

The remittance Information is usually to pass in unstructured form.

6 In this case, it is a foreign payment, usually outside of the SEPA area. All currencies are allowed.

To identify the payment receiver, the IBAN or the bank account number is used.

To address the bank of the payment receiver, there are three options:

V1 Please provide the international BIC of the bank of the payment receiver.
V2 Please provide the bank clearing code and the name and the address of the bank of the payment receiver.
V3 Pease provide the name and the bank of the bank of the payment receiver.

The remittance Information is usually to pass in unstructured form.

19.2 Direct Debit

In Switzerland there are two different direct debit methods, which can performed with XML-files. Technological, these two methods will be shown in SepaTools with three fictitious kinds of payments (in Switzerland not explicit defined).

101 In this case, it is the debit direct method of the postal finance (DD). It is in a wide base identically with the method known in Germany and Austria.

Creditor and debtor are defined with BIC and IBAN. Allowed are the currencies EUR and CHF.

The remittance Information is usually to pass in unstructured form.

This method is prior used in foreign payments into the SEPA area. For direct debit inside of Switzerland, you should use the method 103.

102 This kind of payment is used by the other banks in Switzerland (TA). It differs significant from the known methods in Germany and Austria. Especially there are no mandates used. Also the providing of a sequence (i.e. RCUR) is not allowed.

To identify the account of creditor (payment receiver) the IBAN or the account number of the creditor are used. The bank of the payment receiver is addressed with the bank clearing number.

The same is valid for the bank of the debtor.

The remittance Information is to pass structured. Normally this is an IPI reference number or an ESR reference number. In this context, for the type of the structured remittance Information are to pass **IPI** or **ESR**.

103 Switzerland has in the meantime the direct debit methods of the banks (TA) and the postal finance (DD) harmonized. With the payment method 103, you can create domestic direct debit payments for Switzerland. Mandates are not required. If you use an account number instead of the IBAN, please use the field ChKonto.

Please note, that the XML-file must be presented unmixed, because the XML schemes are very differently. The payment kinds 101, 102 and 103 (postal finance and banks) must not be mixed.

19.3 Plausibility Checks

Because of the different opportunities to address the creditor/debtor, there are no bank specific plausibility checks available. The calling program is responsible that the passed values fit to the format standards.

19.4 Validation of the XML-Files

The created XML-files can be validated by using the Switzerland XSD-files, available by

http://www.six-interbank-clearing.com/de/home/standardization/iso-payments/customer-bank/implementation-guidelines.html.

Additionally to them, the validation portal https://validation.iso-payents.ch/validation/(S(hnzdyp4t0y4mlqjv5dwfxlh3))/KUNDEBANK/login.aspx?Proj=1&Lang=DE is available.

19.5 Implementation

The additional functions for the support of XML-files for banks in Switzerland and Liechtenstein will be provided with the functions SepaTools_SetVersionUndLand and SepaTools_WriteXMLExt. In the following the changes and extensions in this functions are displayed.

20 SepaTools_XMLSetId

20.1 Purpose of the function

Within an XML file, according to the specifications identifiers have to be set. These are unique identifiers in order to identify each single payment-collector (PmInfoId) and individual records (EndToEnd ID).

These identifiers usually are generated internally by the API. In case that you want to put these labels individually, you can use the function SepaTools_XMLSetId.

The call to the function has to be immediately before calling the function SepaTools_WriteXML. The values set by this function are stored in the API global and are saved until there will be set new values.

The sequence presents itself as the following:

- Calling of SepaTools_CreateXML — Initialization of the processing
- Calling of SepaTools_XMLSetId — Setting the identifiers
- Calling of SepaTools_WriteXML — This call can be repeated as often as you want (limited by the available memory of the computer)

  In the structure which is passed, the data from each payment order is passed (per call).

  …
- Calling of SepaTools_CloseXML — The application is closed and the file is written to the specified disk.

20.2 Function call

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

20.3 Parameters

PmInfo

Please pass here a pointer to the identification (ID collectors) which you want to pass here. The identifier can be up to 35 characters. If the identifier contains characters that are not allowed within the XML file, they will be automatically filtered.

If you pass an empty string in this parameter, the identifier is again determined automatically (see section 16.3 on page 47).

Due to the fact that debits are possible with different execution dates or sequences (once, recurrent, etc.), there result internally different collectors. For every collector there has to be specified a unique PmInfoId be.

When via this function a separate identifier is passed, the application is responsible for the correct sorting of data records. Inconsistencies with the internal sorting are possible.

It is strongly recommended to pass an empty string in this parameter. With this, the PmInfoId IDs are internally formed automatically.

EtEInfo Please pass here a cursor to the identifier (single block ID) which you want to pass here. The identifier can be up to 35 characters. If the identifier contains characters that are not allowed within the XML file, they will be automatically filtered.

If you pass an empty string in this parameter, the identifier is again determined automatically (see section 16.3 on the page 47).

21 SepaTools_XMLGetData

21.1 Purpose of the function

Under the condition that an XML file was generated (see page 42) the content of the XML file can be read as a memory map (byte array).

The sequence presents itself as the following:

- Calling of SepaTools_Create — Initialization of the processing
- Calling of SepaTools_WriteXML — This call can be repeated as often as you want (limited by the available main memory of the computer)

  In the structure which is passed, the data from each payment order is passed (per call).

  …
- Calling of SepaTools_CloseXML — The application is closed and the file is written to the disk or the memory.
- Calling of SepaTools_XMLGetData — The contents of the XML file will be retrieved as an array of bytes from the memory.

21.2 Function Call

int SepaTools_XMLGetData(char *Data, int *Size)

21.3 Parameters

Data — After the function call the pointer "data" points to in the memory area, in which the content of the XML file is located. Via the function SepaTools_XMLGetData, the address of the memory area is copied to the variable "Data".

Please note:

The required memory space is allocated by the API. The application is not allowed, respectively doesn't need to allocate own memory space.

The memory area for the data which have to be passed is managed by the API. Each time this function is called a previously allocated memory area is released, before a new memory area is allocated.

Ending the API via a SepaTools_Free, there will be finally released a potential allocated memory.

Size — In this variable, the size of the data is returned. The data is completed with a closing 0 (NULL). Because of this, the value for the size of the memory area which is returned in this variable is one byte greater as the actual data.

21.4 Return codes

0	Everything was successful
-1	To retrieve the content of the XML file as a byte array, it is necessary that there is passed an empty string for the name of the XML file at SepaTools_CreateXML. This is not the case here. To use this function, it is necessary to pass an empty string as the filename for the XML file, at SepaTools_CreateXML.
-2	The memory area for the transfer of data has not been allocated. Possibly the order of function calls is wrong. See page 68.
-999	The API has not been initialized. Please call first to the function SepaTools_Init auf.

22 SepaTools_XMLFreeData

22.1 Purpose of the function

This feature allows to release the memory area, which was allocated by function SepaTools_XMLGetData. As already mentioned, this task hasn't been done mandatory by the API, because the API manages the memory at this point internally.

This function makes sense when working with very large data volumes and storage space has to be saved.

22.2 Function call

int SepaTools_XMLFreeData()

22.3 Parameters

none.

22.4 Return codes

0	Everything was successful.
-1	There was no space available, which would have to be released. This return value is also available if you call this function multiple times without the memory was allocated again in the meantime.
-999	The API has not been initialized. Please call first to the function SepaTools_Init.

23 To convert DTA-files in XML-files

The API can convert DTA files into to XML files. There are also supported multi-DTA files. Multi-DTA files are DTA files which content multiple logical DTA files in one physical file (also mixed between transfers and direct debits).

In this documentation it is often spoken of the difference between logical file and physical file. In the practice there will be mostly only one logical file will be included in the physical file. In this case the logical file has always the value 1.

The basic procedure represents itself as the following.

- Calling of SepaTools_ReadDTA	The physical DTA file is read into a temporary file and if necessary it is internally divided into several logical DTA files.
- Calling of SepaTools_GetDTAFehler	This function call is optional. This feature allows to readout detailed information, in case that while memorizing of the DTA- file content errors (e.g., amount = 0) were observed.
- Calling of SepaTools_GetDTAInfo	If necessary, this function has to be called multiple times. It provides a structure with information from the logical DTA file. Several fields in the structure are not filled for now. These have to be used in the next function.
- Calling of SepaTools_PutDTAInfo	If necessary, this function has to be called multiple times (for each logical file). The structure is delivered out of previous calls to the function SepaTools_GetDTAInfo The content of the structure has to be/can be amended or modified. For details see page 79 of the function description
- Calling of SepaTools_WriteDTAtoXML	This function creates the temporary file from the actual XML file. If in the physical DTA file, there were contented credit transfers and direct debits (multiple logical files), this call (and the following call option) is required twice.
- Calling of SepaTools_XMLGetData	This call is optional and will only be used if the XML file has to be transferred directly into memory.
- Calling of SepaTools_GetDTAProtokoll	With the repeated call of this this function, you can call journal data of the successful and not successful written data.

- Calling of SepaTools_FreeDTAVars — This function releases the internally memory which was allocated for the processing. The temporary files are deleted.

  This is the last function call which is necessary here.

Is established based on the function SepaTools_GetDTAInfo that there have been both credit transfers and direct debits in the DTA file, then the block Sepa Tools_WriteDTAtoXML and if necessary SepaTools_GetDTAProtokoll has to be called twice (once for transfers and for direct debits).

24 SepaTools_ReadDTA

24.1 Purpose of the function

This function is used to read a DTA file. Likewise, the internally required initialization is made.

While reading the DTA file, there are extensive formal and substantive checks. Is there a first unrecoverable error, the function breaks off (there is no automatic error correction possible).

Errors that affect only an individual record, are suspended from further processing. The error-free data records are processed. Information about these data records can be accessed via the function SepaTools_GetDTAFehler.

Optionally, you can evaluate the structure "DTAFehler". This delivers detail information on content errors that prevent further processing.

24.2 Function call

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

24.3 Parameters

Path Please pass a pointer to the full file name (including the path) of the DTA file which has to be read (z.B. C:\Test\DTAUS0.txt).

DTAFehlerStruct Here a pointer is to pass to the structure DTAFehlerStruct. The structure is shown in the following.

In case that by reading the DTA-file errors are recognized and the function aborts with a negative return code, the structure is filled in with detail information.

If you don’t want to evaluate this information, you can deliver a NULL-pointer (NULL or Nil) to the structure.

struct DTAFehlerStruct

{	char	ErrorMsg[80+1]	The problem reported as a clear text.
	char	ErrorField[10+1]	The name of the affected field in the DTA-file [1)].
	int	ErrorRec	The number of the record within the logical file [2)].
	int	NumLogDat	The number of the logical file within the file.
}			

Additional notes:

1) According to the DFÜ-agreement the fields of the DTA file are with a letter (A,C,E) and a following number numbered consecutively (e.g. C7).

2) The number of the record is counted up per E-record, C- record and A- record.

24.4 Return codes

0	Everything was successful
-1	The DTA-file could not be opened.
-2	The DTA-file couldn't be found. Please prove the path and the name of the data.
-3	The opened file is no DTA file.
-4	The named temporary path does not exist (see SepaTools_Init).
-5	The temporary DTA file could not be generated.
-6	Error while writing the temporary file.
-7	The temporary file could not be opened.
-8	Error while reading the temporary file
-9	The DTA file could not be closed successfully
-11	There is an error which can't be corrected in the A record (see DTAFehlerStruct).
-12	There is an error which can't be corrected in the C record (see DTAFehlerStruct).
-13	There is an error which can't be corrected in the E record (see DTAFehlerStruct).
-14	E record was found, although there wasn't found a previous A record.
-15	E record was found, although there wasn't found a previous C record.
-999	The API wasn't initialized. Please first call the function SepaTools_Init.

25 SepaTools_GetDTAFehler

25.1 Purpose of the function

Are there errors while reading the DTA file, which can't be processed correctly, they are suspended from the process.

Such errors can be for example:

- Amount is zero
- Bank code number is smaller than 10000000
- Bank account number is smaller than 1
- Name is missing
- and so on

The call of this function is optional!

When this function is used, it has to be called repeatedly eventually.

25.2 Function call

int SepaTools_GetDTAFehler(DTADetailStruct *DTADetail, int *Index)

25.3 Parameters

DTADetail Please pass a pointer to the structure "DTADetailStruct". If while reading the DTA file errors were found, which allow further processing of the error-free data, so with this function information concerning the errors is emitted.

Index Please deliver the value 1 when you call the function for the first time. This means that the first error entry is returned from the list of detected errors in the structure.

As part of the function call, the variable "index" is automatically incremented to the value 1. In the function call that follows, the next entry from the error list is returned.

Is as an "index", a value less than 1 or a value that is greater than the maximum entries of the error list delivered, the function returns a negative return value (see below) and in the variable "index" the number of actual entries.

By manually setting the variable "index" can also be accessed on the entries in the error list.

struct DTADetailStruct

{	int	LogDat	Number of the logical data
	int	Rec	Number of the record within the logical data.
	char	Feld[5+1]	Number of the field (DTA-Specification [1)])
	char	Fehler[50+1]	Description of the error (text in clear)
	char	Name[27+1]	Name of the recipient of the payment, if available [2)]
	int	Direct debit	Value 1 (direct debits), else 0

char	Betrag[12+1]	Amount of the payment [3)]
char	Reserve[50+1]	Free for later using

}

Additional notes:

1) According to the DTA-specification the description of the fields is according to A-record, C-record and E-record numbered consecutively (e.g. C4 or E5).

2) If the absence of the recipient's name was recognized as an error, in this field no name is returned.

3) If the absence of the amount was recognized as an error, in this field no amount is returned.

25.4 Return codes

0	Everything was successful. There are more data available. Repeated calling (without changing the variable "Index") is required.
-1	The memory area for the error list is not allocated. Probably there hasn't been called the function SepaTools_ReadDTA.
-2	No items were written to the error list. This means, there were no errors found. This should be the norm.
-3	As "index", a value was delivered less than 1. In the variable "index" the number of items in the error list is returned.
-4	As "index", a value was passed, which is larger than the number of available list entries in the error list. In the variable "index" the number of items in the error list is returned.
-5	The entry list is empty. This error should not occur and indicates a programming error.
-999	The API has not been initialized. Please first call to the function SepaTools_Init.

26 SepaTools_GetDTAInfo

26.1 Purpose of the function

After a physical DTA file has been read into a temporary file, information about the logical files can be got with this function. Therefore the function is called repeatedly until the return value is <> 0.

The information out of this function is a base for the following calls of SepaTools_PutDTAInfo.

26.2 Function call

int SepaTools_GetDTAInfo(DTAInfoStruct *DTAInfo, int *First)

26.3 Parameters

DTAInfo Please pass a pointer to the structure „DTAInfoStruct“. The structure will be filled particularly with this call of the function. The structure is delivered for each logical file because of the repeatedly call of the function. The structure is shown as the following.

First When the function is called for the first time, the variable has to be filed with the value of 1. The variable is automatically filled with a 0 after of the first call.

struct DTAInfoStruct

{	int	LogFileOK	This value is placed with [1].
	int	Auftragsart	Order type=0 credit transfer, =1 direct debit
	int	LogDat	Number of the logical file [2]
	char	BLZAuftraggeber[8+1]	Bank code number of the initiator
	char	KontoAuftraggeber[10+1]	Bank account number of the initiator
	char	Auftraggeber[70+1]	Name of the initiator [3]
	char	ErstDatum[8+1]	Creation date of the DTA file (DDMMYYYY).
	char	AusfDatum[8+1]	Date of execution, if available (DDMMYYYY).
	int	NumSatz	Number of logical data records in the logical file
	char	SummeBetrag[12+1]	Sum of amounts in the logical file [4]
	char	SummeKonto[17+1]	Sum of the bank account number in the logical file.
	char	SummeBLZ[17+1]	Sum of the code account number in the logical file
	char	CI[35+1]	Not allocated. See chapter 27.1, page 79.
	char	Mandat[35+1]	Not allocated. See chapter 27.1, page 79.
	char	MandatDat[8+1]	Not allocated. See chapter 27.1, page 79.
	char	Sequence[4+1]	Not allocated. See chapter 27.1, page 79.
	int	AutoMandat	Not allocated. See chapter 27.1, page 79.
	int	MandatStartNr	Not allocated. See chapter 27.1, page 79.
	char	PmInfo[10+1]	Not allocated. See chapter 27.1, page 79.
	char	EToEInfo[10+1]	Not allocated. See chapter 27.1, page 79.
	int	B2B	Not allocated. See chapter 27.1, page 79.
	char	Purpose[4+1]	Not allocated. See chapter 27.1, page 79.
	char	AusfZeit[8+1]	Optional execution time for instant payments
	char	Reserve[32]	
}			

Additional notes:

1) This value is set to 1 automatically while reading the physical file. If you don't want to include this logical file in the conversion, you have to set this value to 0 before calling SepaTools_PutDTAInfo.

2) This is the number of the logical file within the read physical file.

   Attention:

   This value must not be changed, because with this value used to control access to internal data.

3) Although in the DTA set, the name of the initiator has a length of max. 27 characters, in the structure 70 characters are available. Before calling the function SepaTools_PutDTAInfo you can change the name of the initiator of the payment and you have 70 characters available.

4) Amounts are always represented as a string in cents. There is no decimal point or thousands separators. Example: 1.342,76 € is shown as the 134276

26.4 Return codes

0	Everything was successful. There are more data available. Repeated calling (with First parameter = 0) is required.
-1	There is no information about logical files at a DTA file. Probably the function SepaTools_ReadDTA has not been called.
-2	In the physical DTA file no logical DTA files were available. Please check the DTA file.
-3	You have reached the end of the data. There are no more data on logical DTA files available.
-4	There appeared a logical memory error (should never happen). Please inform the manufacturer.
-999	The API has not been initialized. Please first call to the function SepaTools_Init.

27 SepaTools_PutDTAInfo

27.1 Purpose of the function

After the information of the logical files could bean read out about the function Sepa-Tools_GetDTAInfo there is the possibility (or you have) to modify the structure and write back with this function.

Particularly if you want to convert direct debits, these for direct debits relevant fields have to be filled in.

You have to call this function for each recognized logical file.

27.2 Function call

int SepaTools_PutDTAInfo(DTAInfoStruct *DTAInfo)

27.3 Parameters

DTAInfo Please pass a pointer to the structure „DTAInfoStruct“. You received the structure because of the calls of the function SepaTools_GetDTAInfo.

The structure is shown as the following.

struct DTAInfoStruct

{	int	LogFileOK	This value is placed with [1].
	int	Auftragsart	Order type=0 credit transfer, =1 direct debit
	int	LogDat	Number of the logical file [2]
	char	BLZAuftraggeber[8+1]	Bank code number of the initiator
	char	KontoAuftraggeber[10+1]	Bank account number of the initiator
	char	Auftraggeber[70+1]	Name of the initiator [3]
	char	ErstDatum[8+1]	Creation date of the DTA file (DDMMYYYY)
	char	AusfDatum[8+1]	Date of execution, if available (DDMMYYYY)
	int	NumSatz	Number of logical data records in the logical file
	char	SummeBetrag[12+1]	Sum of Amounts in the logical file [4]
	char	SummeKonto[17+1]	Sum of the bank account number in the logical file
	char	SummeBLZ[17+1]	Sum of the code account number in the logical file
	char	CI[35+1]	CI (Creditor identification) for direct debits [6]
	char	Mandat[35+1]	Mandate code for direct debits. [7]
	char	MandatDat[8+1]	Date of the mandate [7]
	char	Sequence[4+1]	Frequency of the execution for direct debits [7]
	int	AutoMandat	Automatically mandates numbering [8]
	int	MandatStartNr	Start of automatically mandates numbering [8]
	char	PmInfo[10+1]	Code for the collector information [9]
	char	EToEInfo[10+1]	Code for the single record information[10]
	int	B2B	Kind of the direct debit [12].
	char	Purpose[4+1]	Purpose code.
	char	Reserve[46]	Place for later use
}			

Additional notes:

1) This value is set to 1 for every logical file automatically while reading the physical file. If you don't want to include this logical file in the conversion, you have to set this value to 0 before calling SepaTools_PutDTAInfo.

2) Ether this is not wise, you are able to change the type of payment when you change this character. So you are able to change credit transfers in direct debits and the other way round.

3) This is the number of the logical file within the read physical file.

   Attention:

   This value must not be changed, because with this value used to control access to internal data.

3) Although in the DTA set, the name of the initiator has a length of max. 27 signs, in the structure 70 characters are available. Before calling the function SepaTools_PutDTAInfo you can change the name of the initiator of the payment and you have 70 characters available.

4) Amounts are always represented as a string in cents. There is no decimal point or thousands separators. Example: 1.342,76 € is shown as the 134276

6) For direct debits, there has to be stringently delivered the creditor identification (CI) of the recipient of the payment. The CI has to be requested at the Deutschen Bundesbank (www.bundesbank.de). To test, you can use CI DE98ZZZ09999999999 .

7) Within the description for the data records for DTA files, there are no fields for information concerning mandates available. This problem can be solved with a help solution.

   Information concerning mandates can be identified, if they look like the following.

   //MID mandate reference

   The code //MID (upper and lower case doesn't matter) says that the following text is a mandates reference. The text which is following to the code is seen as a mandate

   //MDAT 17.04.2011

   The code //MDAT (upper and lower case doesn't matter) says that the following date is the date from which on the mandate is valid. The date (e.g. 17.04.2011) can have the following dimensions.

DD.MM.YYYY	17.04.2011
DD.MM.YY	17.04.11
DDMMYYYY	17042011
DDMMYY	170411

   //SEQ FRST

   With the code //SEQ you are able to name the rhythm of direct debits. Valid values are:

FRST	for the first direct debit
RCUR	for a recurring direct debit (this will be the norm)
OOFF	for a non-recurring direct debit

FNAL for the last direct debit

The values can be filled in the following fields of a DTA file:

Reason for payment row 1 to 14
2. field for the recipient
2. field for the initiator

An entry in the recipients file can look like the following for a mandate for example.

Example for special information:

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

After the mandates information (special information) have been identified, these will be deleted in the appropriate fields in the DTA file.

If from the DTA file no mandate information can be used, the code for the mandate and the date of the mandate from the structure is used. For an unique code of the mandate, please note the explanation of point 8.

8) With the value „AutoMandat“, you define if and in which way a numeration of the mandate (built on the code of the mandate) will be effected. The following values are available for the value „AutoMandat”:

0 The code of the mandates are not numbered consecutively.

1 The code of the mandates are numbered consecutively (z.B. TestMandat123). The artificial generated mandate will only be used if there is no mandate available.

2 The code of the mandates are numbered consecutively (z.B. TestMandat123). The artificial generated mandate will be used, when there is a mandate available, the available mandate will be overwritten.

The value „MandatStartNr“ defines, from which value the numeration starts.

9) For each collector within the XML file you have to use a unique ID. This ID is generated automatically while generating the XML-File.

You are able to define a 10 numbers long code, which will be included in the generation of the ID. If you don’t define a code, the ID will be generated with previous defined values.

10) For each data record within the XML a unique EndToEnd-Id has to be used. While the creation of the XML-file, the EndToEnd-Id will be created automatically.

You are able to define a code, which is up to 10 characters long. This code will be included in the building of the ID. If you don’t define such a code, the EndToEnd-Id will be built with previous defined values.

11) The date of execution for direct debits has to be in the future (see SEPA Rulebook). For credit transfers, the execution date is may not be in the past.

In case that the date of execution is not valid, the API will define a valid date of execution.

12) To regulate the kind of direct debits, you have to set the variables „B2B“.

The following valued are possible.

0 Normal CORE-direct debit with term of presenting 3 or 6 days.
1 Business-to-Business direct debit B2B. This accords to the previously pre-authorized direct deb its.

2 Direct debit (COR1) with shortened term of presenting of 2 days. The shortened term of presenting is valid for both, for first and recurrent direct debits.

You only are allowed to set the value of 2, if the version is set to 2.7 at least, by the function SepaTools_SetVersionUndLand. If this minimum version wasn't set, the value of 2 (if delivered) is internally set to the value of 0.

The type of direct debit COR1 has to be arranged between the credit institutions bilateral. In Germany and Austria, this is arranged.

27.4 Return codes

0	Everything was successful. There are more data available. Repeated calling (with First parameter = 0) is required.
-1	There is no information about logical files from a DTA file. Probably the function SepaTools_ReadDTA has not been called.
-2	In the physical DTA file no logical DTA files were available. Please check the DTA file.
-3	The delivered value for the LogDat is not in the space, for which data are available. This return code only can appear, if you changed the value for the LogDat.
-4	There appeared a logical memory error (should never happen). Please inform the manufacturer.
-999	The API has not been initialized. Please first call to the function SepaTools_Init.

28 SepaTools_WriteDTAtoXML

28.1 Purpose of the function

With this function, the XML-file will be created, out of the files, which have been created until now. While the XML-file is created, the data records will be sorted and maybe grouped. In Case of grouping the value of the PMInfoId (collector code) will be created (see chapter 27.3 page 79).

28.2 Function call

int SepaTools_WriteDTAtoXML(CreateStruct *XMLCreateStruct)

28.3 Parameters

CreateStruct Here a pointer is delivered to the structure CreateStruct. The structure is shown below. All strings are delivered null-terminated. Because of this, the length of the character arrays is always one character longer than the origin characters string.

All strings chains are proofed internally, if they are a valid characters record, for SEPA payments. If not, the invalid characters are removed. The corrected string will be returned in the structure.

In this function, it is proofed, if the file could be written (writeable disk).

struct XMLCreateStruct

{	char	ExportPfad[255+1]	Drive an path for writing the XML-file.
	int	Direct debit	Credit transfer (=0) or direct debit (=1).
	char	XMLName[35+1]	The name of the XML-file, which has to be created [1)].
	char	MsgId[35+1]	Message-Id (Code) for the XML-file [2)].
	char	EinreicherName[70+1]	Name of the presenter of the file (mind. 3 characters).
}			

Additional notes:

1) Irrespective of the delivered or not delivered expansion of the file name, the expansion of the file name will always be set to .xml.

   If here, an empty string (or the value Dummy.txt) is delivered, the API will manage the file name internally. This is necessary, if the XML-file should be created as a Byte-Array in the memory (See chapter 14.3 page 42).

2) The Message-Id is seen as a nearly unique identification of the XML-file. If you deliver an empty string here, there will be created a Message ID internally and this will be returned in the structure.

28.4 Return codes

0	Everything was successful Note: The function also returns the return code of 0, if all data records were functional faulty. This can happen, if there is given a bank routing number for the initiator, which has a BIC, which isn't available for SEPA. Because of this, the data records are functional faulty, but the technical process was alight. You can prove this, if you call the function GetDTAProtokoll repeatedly.
-1	There was delivered a path for the export file, which was formal wrong (length < 2 characters).
-2	The directory for the export file doesn't exist.
-3	The disk, which was stated in the name of the path of the export file is write-protected. The file is not able to be written.
-4	The name of the presenter of the file was stated too short (< 3 characters) .
-6	The path name for the export file is to long (>200 characters).
-7	The stated directory for the export file is not ready. Maybe there is no data carrier available.
-8	The name for the XML-file was stated too short (< 3 characters).
-9	There is no information available about logical files of a DTA file. Probably the function SepaTools_ReadDTA wasn't called yet.
-10	In the physical DTA-file, there were no logical DTA-files available. Please proof the DTA-file.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

29 SepaTools_GetDTAProtokoll

29.1 Purpose of the function

With this function you are able to read out and protocol the error-free converted and also the faulty converted data records.

For this you have to call the function repeatedly, until the return code has a value of <>0. With the parameter error you can define, if the valid or the invalid data record should be emitted.

29.2 Function call

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

29.3 Parameters

ReadStruct This structure is filled with the data of the payment order with each call of the function.

The structure is shown below:

Fehler If you want to read out the error-free data records, put the parameter „Fehler“ to 0. If you set the parameter „Fehler“ to 1, the faulty, not converted data records are emitted.

First Please put this value to 1, when you call this function for the first time. Because of this, initialization values are set.

When the function is called for the first time, the value is set to 0, by the API. This value (0) has to be delivered for following function calls (until the return code is <>0).

struct XMLReadStruct

{	char	PmInfoId[35+1]	PaymentInfoId – Identification of the order
	char	AusfDatum[8+1]	Date of execution of the payment (form DDMMYYYY).
	char	AuftragName[70+1]	Name of the initiator
	char	AuftragBIC[11+1]	BIC of the initiator
	char	AuftragIBAN[35+1]	IBAN of the initiator
	char	AuftragAbwName[70+1]	Optional a differing name of the initiator
	char	AuftragCI[35+1]	CI of the initiator (only for direct debits)
	char	EmpfName[70+1]	Name of the recipient of the payment
	char	EmpfBIC[11+1]	BIC of the recipient
	char	EmpfIBAN[35+1]	IBAN of the recipient
	char	EmpfAbwName[70+1]	Optional a differing name of the recipient
	char	Purpose[4+1]	Optional you can define a purpose of the payment
	char	Betrag[12+1]	Amount of the payment in Cent.
	char	EndToEndId[35+1]	Code of the individual payment.
	char	MandatId[35+1]	Mandates code (only for direct debits)
	char	MandatDatum[8+1]	Date of the mandate (form DDMMYYYY)
	char	SequenceType[4+1]	Sequence of the direct debit
	int	B2B	B2B-direct debit (<>0) or direct debit (=0).
	int	SammlerAnzahl	Number of payment orders in this collector

```
    char    SammlerSumme[12+1]   Sum of the amount of this collectors in sum.
    char    Zweck1[70+1]         Reason for payment row 1
    char    Zweck2[70+1]         Reason for payment row 2
    int     Hinweis[30]          A field to the point of 30 note codes1)
}
```

Additional notes:

1) By using the structure XMLReadStruct, only the first value within the 30 characters long note field is filled. Ether with >=0 (error-free) or with an error value out of the following list (Detail-error-codes):

29.4 Detail Error Codes

Note:

If return code >=0 (no error), in the note field additional information is saved.

This information is stored in bits.

The occupancy with bits for the value >= 0 in the notes field have the following meaning.

0 Everything was successful

1 The calculated IBAN for the initiator can be used. But it isn't unique. We advise you to contact your client.

This positive return code can only appear when it is a 7 characters long bank account number from the Deutsche Bank.

2 The calculated IBAN for the recipient of the payment can be used. But it is not unique. We advise to contact your client.

From here on the error codes have to be evaluated as negative integer-values.

-104 The Amount of the payment is 0.

-105 The bank of the recipient can't be reached by SEPA.

-110 For the combination bank routing number and bank account number of the recipient of the payment, there can't be calculated an IBAN.

-111 For the combination bank routing number and bank account number of the recipient of the payment, there can't be calculated an unique IBAN.

-112 The bank account number of the recipient of the payment is invalid (check digit is wrong).

-113 The bank routing number of the recipient is flagged for deleting.

-155 The bank of the initiator is not available by SEPA.

-159 The delivered CI (Creditor Identification) is not valid.

-165 For the combination bank routing number and bank account number of the client of the payment, there can't be calculated an IBAN.

-166 For the combination bank routing number and bank account number of the initiator of the payment, there can't be calculated an unique IBAN.

-167 The bank account number of the initiator of the payment is invalid (check digit is wrong).

-168 The bank routing number of the client is flagged for deleting.

-169 The value of the frequency of the direct debit is invalid. Valid are the following values:

FRST	for the first direct debit
RCUR	for a recurring direct debit (this will be the norm)
OOFF	for a non-recurring direct debit
FNAL	for the last direct debit

29.5 Return codes

0 Everything was successful. There are more data available. Repeated calling (with First parameter = 0 before the next call) is required.

-1 There is no information about the converted DTA-files available. Probably the function SepaTools_ReadDTA hasn't been called yet.

-2 In the physical DTA file no logical DTA files were available. Please check the DTA file.

-3 The temporary file couldn't be opened for reading.

-4 You reached the end of the data. There are no more data available.

-999 The API has not been initialized. Please first call to the function SepaTools_Init.

30 SepaTools_FreeDTAVars

30.1 Purpose of the function

The function releases the memory area, which has been allocated by the API for the converting of the DTA files. Temporary generated files are deleted.

int SepaTools_FreeDTAVars()

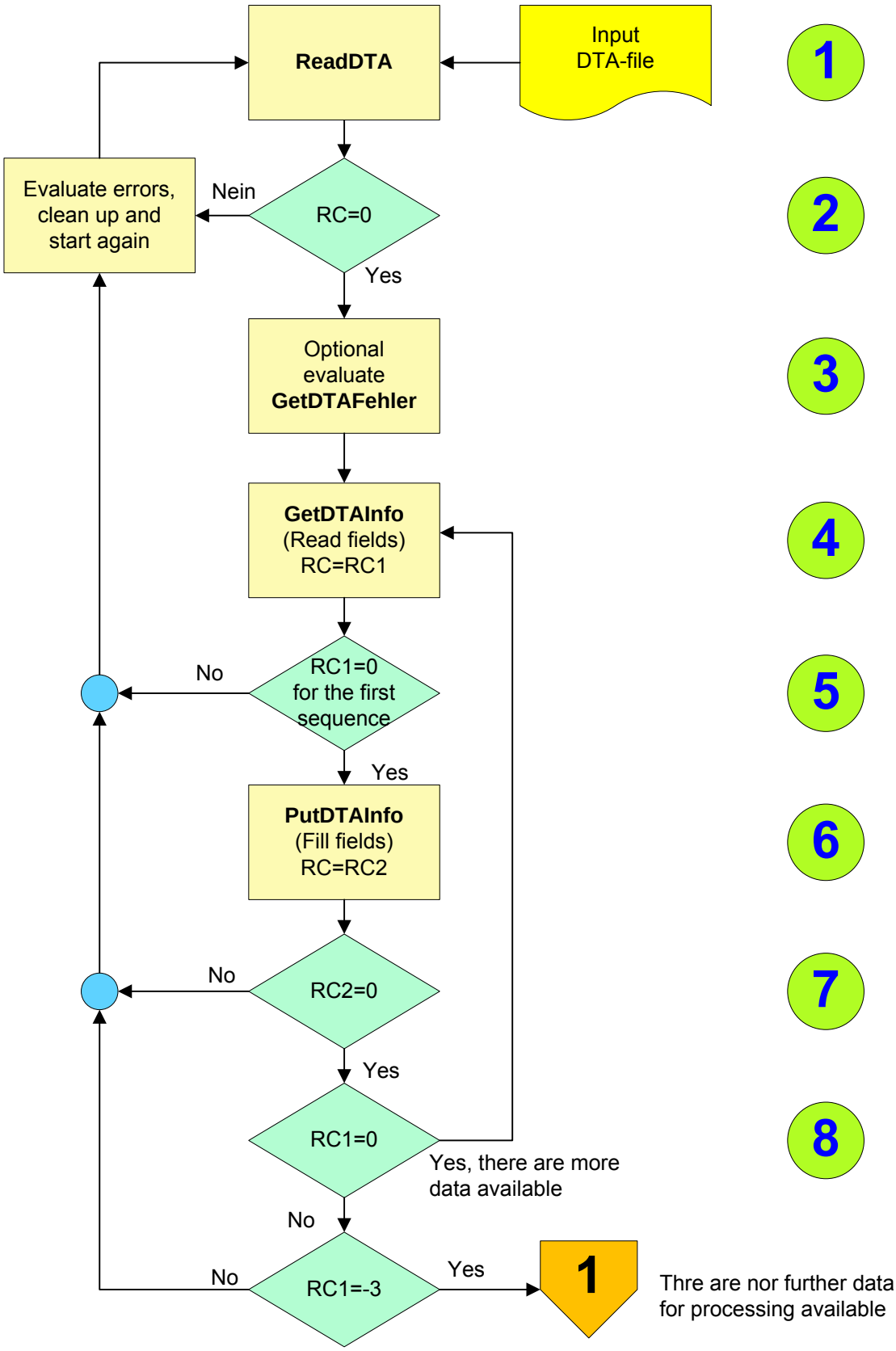
30.2 Parameters

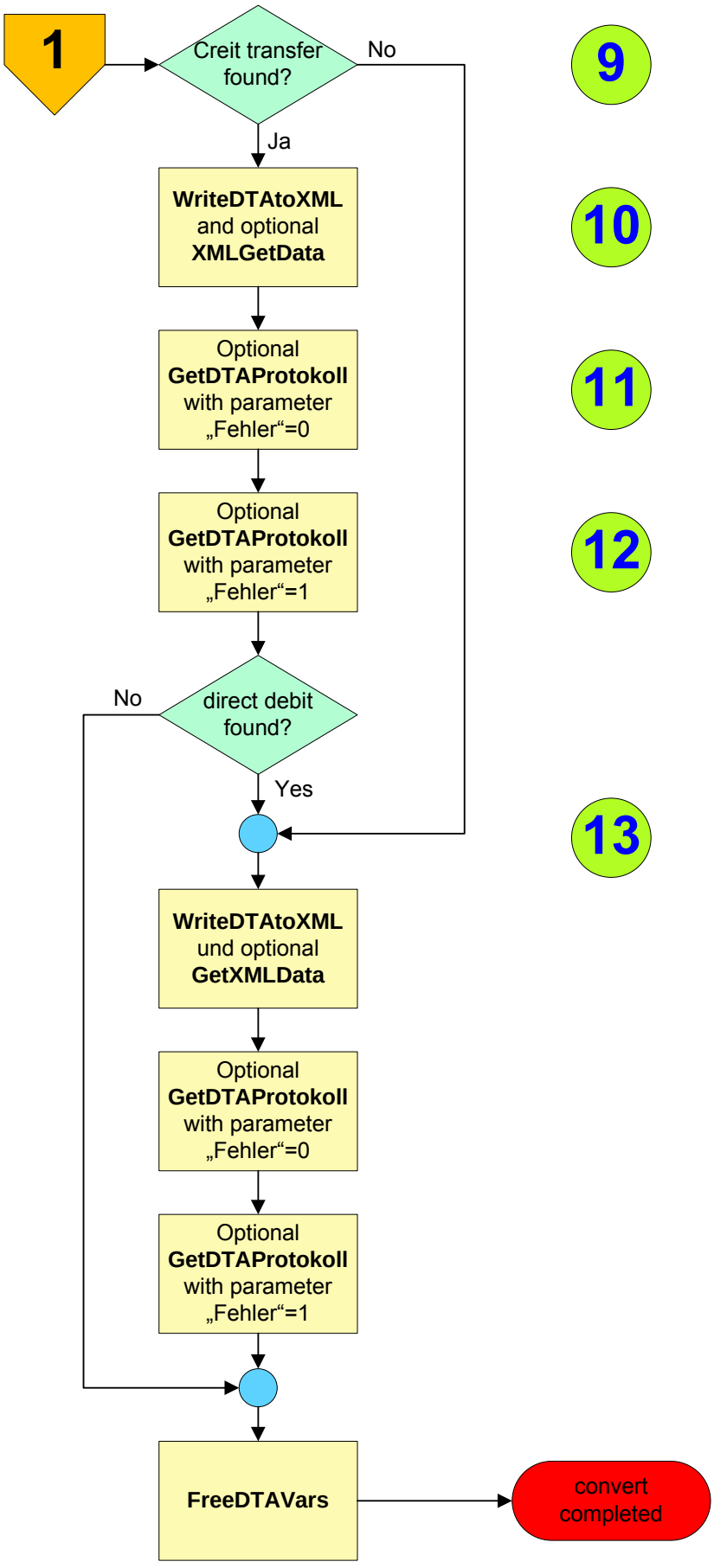
none

30.3 Return codes

0	Everything was successful.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

30.4 Flow chart




1
Creit transfer found?
No
Ja
WriteDTAtoXML and optional XMLGetData
Optional GetDTAProtokoll with parameter „Fehler“=0
Optional GetDTAProtokoll with parameter „Fehler“=1
direct debit found?
No
Yes
WriteDTAtoXML und optional GetXMLData
Optional GetDTAProtokoll with parameter „Fehler“=0
Optional GetDTAProtokoll with parameter „Fehler“=1
FreeDTAVars
convert completed
9
10
11
12
13

30.5 Description of the flow chart

In the following part, the text characters (green circle with blue number) for the flow diagram for converting of DTA files into SEPA XML files are described.

1 Starting base is a DTA-file. It is also possible that this is a multi-DTA-file, in which there are more logical files in a physical file.

Even mostly there is only one logical file in a physical file it is always emanated from a multi-DTA-file.

If there appear errors while reading the DTA-file, which prevent a further processing, in this case the function aborts with a negative return code.

If there appear faulty data records which don't prevent a further processing of the DTA-file (e.g. Amount=0 or BLZ has not 8 characters), only the faulty data records are suspended of the further processing.

2 A negative return code prevents the further processing

3 If there appear faulty data records which don't prevent a further processing, you have the possibility to call these data records (optional) with a repeatedly call of the function „GetDTAFehler".

There is no possibility to heal the errors themselves. The information you get by the function „GetDTAFehler", can help you to correct future DTA-files.

4 By the function „GetDTAInfo", information of the logical file within the physical DTA file are read. In this there are also the sum from the E-record (Number of records , sum of the amounts, BLZ and bank account number) included.

Please note, these are the amounts of the E-records. Differences based on faulty data records are not includes here.

This function has to be called repeatedly, if there is the possibility for more than one logical file in the physical file.

5 You need min. one logical file. Because of this, you get the return code 0, if the call of the function was successful. If this doesn't happen, an error was recognized, which prevents a further processing.

6 At least concerning direct debits, the structure which is delivered to the function „PutDTAInfo" should be enhanced with further information. Essential is the field date of execution. Here the execution term for direct debit has to be considered.

Concerning direct debits, it is proofed, if the date of execution equates at least to the execution term for recurring direct debits (current todays date+3 days with consideration of the Target days). If the delivered execution date is smaller as this date, the calculated date is used.

7 Respective the return codes please consider the documentation.

8 If there are more logical DTA-files in the physical DTA-file, please repeat the call of the sequence „GetDTAInfo“ and „PutDTAInfo“ until the return code of the function „GetDTAInfo“ has a value of <>0 (-3 means that there are no more data available).

9 It is possible that there are credit transfers and direct debits together in in the DTA-file. These are different XML files. This is the reason why the emission of the XML-file has to be twice

10 With the function „WriteDTAtoXML“ the real XML-file is generated out of the previous generated data.

In the structure, which is delivered, there is also the path and the name of the file for the XML-file delivered. If there is passed an empty string for the path and the name of the file, the emission of the XML file happens in a Byte-Array instead of a file in the memory.

Optional, the data can read out by the function „XMLGetData”. Please note chapter 0 page 68 of the documentation.

11 Optional, you can call the function „GetDTAProtokoll“. In the documented structure all data records, which were written in the XML-file successfully are returned. Please deliver the value of 0 by the parameter „Fehler“.

12 To protocol the data records, which are faulty and not written in the XML-file (e.g. the bank is not available by SEPA), you can call to the function „GetDTAProtokoll“ (optional).

Please note: These are not the same errors, which are established by the function „GetDTAFehler“. Technical wrong data records have been sorted out there yet.

Functional wrong data records are sorted about by the function „GetDTAFehler“. E.g. bank routing number is deleted, bank is not available by SEPA, test digit of the bank account number is wrong, and so on…

13 The sequence which is displayed in the text characters 10 to 12 has to be called for a second time.

31 Convert DTAZV-files in XML-files

With the API you are able to convert DTAZV-files (German foreign payment transactions) in XML-files. With this, the data records for the SEPA area can be forwarded cheaper.

The proceeding is more easily compared to DTA-files, because here exist no direct debits and no multi-DTA-files.

Because of the fact that in a DTAZV-file can be included data records, which can't be converted in a SEPA Payment, those are excepted of the conversion. Those are provided in a DTAZV- file (reduced).

The basic process is as the following:

- Call of SepaTools_ConvertDTAZVtoXML	The physical DTAZV-file is read in. With this the SEPA compatible payments are written in a XML-file. Payments which are not SEPA compatible (e.g.: currency no EUR, no SEPA-country) are written in a second DTAZV-file. Those can be processed as before
- Call of SepaTools_GetDTAProtokoll	With the repeatedly call of the function, log data of the successful and not successful written data records can be called. This is the same function which is also used by the conversion of DTA-files. Only those data records are emitted as error-free, which were written into the XML-file. All the other data records, which were written in the DTAZV-file are logged as faulty data records.

32 SepaTools_ConvertDTAZVtoXML

32.1 Purpose of the function

The function reads out the DTAZV-file and converts SEPA-compatible data records into SEPA-XML-files. Data records with are not SEPA-compatible are written in an additional DTAZV file.

32.2 Call of the function

int SepaTools_ConvertDTAZVtoXML(const AZVConvertStruct *AZVConvert)

32.3 Parameters

AZVConvert By this parameter, all necessary variables are delivered to the function by a pointer to a memory area.

The structure is shown in the following.

struct AZVConvertStruct

{	char	InputFile[255+1]	The path and the name of the file for the DTAZV-file
	char	OutputFile[255+1]	The path and the name of the file of the XML-file which has to be generated.
	char	AZVFile[255+1]	The path and the name of the file for the additional DTAZV-file [1)]
	char	AusfDatum[8+1]	The date of execution for the SEPA payments [2)]
	char	Kennung[10+1]	A code for the identifier in the XML-file [3)]
	char	Reserve[200]	Buffer for later use
}			

Additional explanation:

1) An additional DTAZV-file is only generated, If there can be found data records which couldn't be converted into SEPA-data records.

2) In the DTAZV-file is included a date of execution as well. If you name here a date of execution which is higher than the date of execution in the DTAZV-file, this date will be used.

   If you don't name a date or you name a smaller date, in this cases the date of execution of the DTAZV-file (InputFile) is used.

   Is the date of execution in the DTAZV-file which has to be converted invalid and you didn't name a date here, there will be used the date of current day + 1 day.

3) To characterize the individual data records within the XML-file identifiers are used. You are able to name an up to 10 characters long code, which is used for the creation of the identifiers. The identifier is completed internally with further information

   If you don't name anything, internally the value of "Kennung" is used for the code.

32.4 Returncodes

0 Everything was successful.

1 Everything was successful. There couldn't be found any SEPA compatible payments. The additional DTAZV-file was generated.

-1 The stated DTAZV-file couldn't be opened, respectively couldn't be found.

-2 The path of the output file (the XML-file which has to be generated) is formal invalid (less than 2 characters).

-3 The declaration of the path (with file name) for the output file (XML-file) is too long (more than 200 characters).

-4 The drive in the path for the output file is not ready.

-5 The directory which was named for the output file doesn't exist.

-6 The data carrier, which was named for the output file is write protected.

-7 The path for the AZV-file (the additional generated DTAZV-file) is formal invalid (less than 2 characters).

-8 The declaration of the path (with file name) for the DTAZV-file which has to be generated additionally is too long (more than 200 characters).

-9 The directory in the path for the additionally generated DTAZV-file is not ready.

-10 The directory, which was named for the DTAZV-file which has to be generated additionally doesn't exist.

-11 The data carrier, which was named for the DTAZV-file which has to be generated additionally is write protected.

-21 The temporary file couldn't (physical) not be generated.

-22 This is no valid DTAZV-file (Incoming file).

-23 In the temporary file could not be written (literal error).

-24 The bank code number of the initiator is invalid. It couldn't be found in the stock of data of the Deutsche Bundesbank.

-25 The bank account number of the initiator is invalid. The calculation of the check digit is failed.

-26 The bank connection of the initiator can't be converted in BIC and IBAN clearly. Please ask your bank for further information.

-999 The API wasn't initialized yet. Please first call the function SepaTools_Init.

33 SepaTools_GetDTAProtokoll

33.1 Purpose of the function

With this function you are able to read out the error free converted data records and also the faulty data records and also log it. This is the same function which is also used when converting DTA-files.

The function has to be called multiple times until the returncode has the value of <>0. With the parameter “Fehler” you regulate if the valid or the faulty data records are emitted.

33.2 call of the function

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

33.3 Parameters

Please have a look to the original description of this function in the part of the conversion of DTA files.

33.4 Detail-Error-codes

Please have a look to the original description of this function in the part of the conversion of DTA files.

33.5 Returncodes

0	Everything was successful. There are more data available. You have to call the function repeatedly (with parameter First=0).
-3	the temporary file can’t be opened for reading.
-4	You have reached the end of the data base. There are no more data available.
-999	The API wasn’t initialized yet. Please first call the function SepaTools_Init.

34 To convert CSV-files into XML-files

CSV-files can be converted into SEPA XML-files by the API. With this flexible solution you are able to generate payment orders with Excel-charts (Exported as a CSV-file).

This allows a flexible triggering of information with the structure which has to be delivered.

The basic process is as the following:

- Call of SepaTools_ConvertCSVtoXML	The CSV-file is read in. With the entries of the individual fields, the payment orders are built and a SEPA XML-file is generated.

- Call of SepaTools_GetDTAProtokoll	With the repeatedly call of this function log data of the successful and not successful written data records can be received. This is again the function which is famous of the DTA-conversion.

35 SepaTools_ConvertCSVtoXML

35.1 Purpose of the function

The function reads out the CSV-file and converts SEPA-compatible data records into SEPA-XML-data.

35.2 Call of the function

int SepaTools_ConvertCSVtoXML(const CSVConvertStruct *CSVConvert)

35.3 Parameters

CSVConvert By this parameter, all necessary variables are delivered to the function by a pointer to a memory area.

The structure is shown in the following.

struct CSVConvertStruct

{	char	InputFile[255+1]	The path and the name of the file for the CSV-file.
	char	OutputFile[255+1]	The path and the name of the file for the XML-file which has to be generated.
	char	AusfDatum[8+1]	The date of execution for SEPA payments [1)]
	char	Kennung[10+1]	A code for the identifiers in the XML-file [2)]
	int	Lastschrift	Value=0 for credit transfer, value=1 for direct debits
	int	B2B	Value for the kind of direct debits [3)]
	char	Einreicher[70+1]	Optionally name of the presenter of the XML-file [4)]
	char	AuftragName[70+1]	Optionally name of the initiator of the payment orders. [5)]
	char	AuftragBLZ[8+1]	Optionally bank code number of the initator[6)]
	char	AuftragKonto[11+1]	Optionally bank account number of the initiator [7)]
	char	AuftragBIC[11+1]	Optionally BIC of the initiator [8)]
	char	AuftragIBAN[35+1]	Optionally IBAN of the initiator [9)]
	char	CI[35+1]	Optionally CI of the initiator [10)]
	char	Mandat[35+1]	Optionally mandate, only with direct debits [11)]
	char	MandatDat[8+1]	Optionally the date of the mandate, only with direct debits [12)]
	int	MandatStart	Opt. start number for numbering of the mandate [13)]
	int	CSVFeld[60]	Field for the positions of the elements within the CSV-file [14)]
	char	Trennzeichen[1+1]	Optionally the divider for the elements [15)]
	char	Grenze[1+1]	Optionally the sign for the field-border definition [16)]
	char	Tausend[1+1]	Optionally the sign for the thousand-separation [17)]
	char	Komma[1+1]	Opt. the sign for the comma representation with amounts. [18)]
	int	Headerzeilen	Opt. Header lines to skip [19)]
	char	AusfZeit[8+1]	Optional execution time for instant payments
	char	Reserve[187]	Reserved for later use
}			

Additional explanation:

1) Please state here the date of execution for the payment with the form of DDMMYYYY. Internally it is proofed, if the date of execution for the chosen payment is valid. There are also considered weekends and holidays.

Respective direct debits, the presenting terms are considered. If the date is not late enough, a valid date is calculated.

2) To characterize the individual data records within the XML-file identifiers are used. You are able to name an up to 10 characters long code, which is used for the creation of the identifiers. The identifier is completed internally with further information

   If you don't name anything, internally the value of "Kennung" is used for the code.

3) Please state here, which kind of direct debits you want to construct. Valid Values are:

   0 Standard-direct debit with presenting terms of 3 days (recurring direct debits) and 6 days (first time direct debits).

   1 B2B – Business direct debit with a presenting term of 2 days. This direct debit may only be used by companies (non-customers). The mandate has to be deposited at the bank of the payer.

   2 COR1 – direct debit with shortened terms of presenting of 2 days. This kind of direct debits is no SEPA-Standard and has to be agreed between the credit institutes. In Germany and in Austria, those agreements are concluded by November 2013.

     Please ask you bank if COR1 direct debits are possible.

4) Please state here optionally the name of the presenter of the XML-file. Basically in one XML-file, there can be included payment orders of different initiators.

   If you don't state a name of the presenter here, there is internally the first name of an initiator, of a payment which is found, used for the name of the presenter.

5) Basically it is possible to deliver the name of the initiator of the payment in a field within the CSV file. Is there no name of the initiator delivered in the CSV file it will be token the stated name.

6) Basically the function is able to handle bank code numbers, bank account numbers and also BIC and IBAN. The state of these values in the CSV file is alternative.

   If there couldn't be found a valid combination out of BIC and IBAN, it will be tried to determine IBAN and BIC out of bank code number and bank account number. If this is also not possible, the data record will be refused.

   If there is delivered no bank code number for the initiator within the CSV file, in this case the here optional stated bank code number is used.

7) For the bank account number of the initiator the facts of item 6 are true.

8) For the BIC of the initiator the facts of item 6 are true.

9) For the IBAN of the initiator the facts of item 6 are true.

10) The CI (Creditor Identification) can also be delivered within the CSV file. Optionally it can be stated here. In this case this CI will be used.

Please note, that if you deliver the CI within the CSV file, you also have to change the name of the initiator.

11) The mandate for a direct debit should if possible always be delivered within the CSV file, because it is an individual mandate of a payer.

Only if this is not possible, with this value will be generated a mandate ID which will be completed with a numbered value.

12 Also the date of the mandate should always be delivered within the CSV file. If this is not possible there can be delivered a date with the format DDMMYYYY. This date is used for all mandates, which have no valid date within the CSV-file.

13 This is the starting number which numbers the mandate (corresponding to item 11).

14 In this field (30 characters) you define on which position of the CSV file the corresponding field contents can be found. The following definitions are in declared:

CSVFeld[1]	Please state here the position of the name of the initiator within the CSV file (e.g.: the value of 3, if the name of the initiator is the third value in the row of the CSV file). If you don't want to deliver the name of the initiator and take out of the structure (see item 5), please set this entry with the value of 0.
CSVFeld[2]	Bank code number of the initiator. Else the facts above are valid.
CSVFeld[3]	Bank identifier code of the initiator. Else the facts above are valid.
CSVFeld[4]	BIC of the initiator. Else the facts above are valid.
CSVFeld[5]	IBAN of the initiator. Else the facts above are valid
CSVFeld[6]	Name of the recipient of the payment. Else the facts above are valid.
CSVFeld[7]	Bank code number of the recipient of the payment. Else the facts above are valid
CSVFeld[8]	Bank account number of the recipient of the payment. Else the facts above are valid.
CSVFeld[9]	BIC of the recipient of the payment. Else the facts above are valid
CSVFeld[10]	IBAN of the recipient of the payment. Else the facts above are valid.
CSVFeld[11]	Amount of the payment. Else the facts above are valid
CSVFeld[12]	Reason for payment of the payment. Else the facts above are valid.
CSVFeld[13]	CI (Creditor identification) of the initiator. Else the facts above are valid des
CSVFeld[14]	Sequence of the direct debit. If there is no value delivered, internally the value RCUR (recurrent direct debit) is set.

CSVFeld[15] Mandate for the direct debit. Else the facts above are valid.

CSVFeld[16] Date of the mandate. Else the facts above are valid.

CSVFeld[17] Not used.

CSVFeld[18] Original mandate (before changed) . Else the facts above are valid.

CSVFeld[19] Original name of the creditor (before changed) . Else the facts above are valid.

CSVFeld[20] Original CI of the creditor (before changed) . Else the facts above are valid.

CSVFeld[21] Original IBAN of the debitor (before changed) . Else the facts above are valid.

CSVFeld[22] SMDA (**S**ame **M**andat with **N**ew **D**ebtor **A**gent). See description of then function WriteXMLExt. Else the facts above are valid.

CSVFeld[23] Different client. Else the facts above are valid.

CSVFeld[24] Different recipient. Else the facts above are valid.

CSVFeld[25] Purpose code. Else the facts above are valid.

CSVFeld[26] Execution date. Else the facts above are valid.

CSVFeld[27] EndToEnd-ID. This value overwrites the automatically generated EndToEnd-ID. Else the facts above are valid.

CSVFeld[28] to CSVFeld[34] Reserved for later use.

CSVFeld[35] Please place here the position of the Country code of the initiator of the payment.

CSVFeld[36] Please place here the position of the address line 1 of the initiator of the payment.

CSVFeld[37] Please place here the position of the address line 2 of the initiator of the payment.

CSVFeld[38] Please place here the position of the Country code of the recipient of the payment.

CSVFeld[39] Please place here the position of the address line 1 of the recipient of the payment.

CSVFeld[40] Please place here the position of the address line 2 of the recipient of the payment.

CSVFeld[41] Please place here the position of an optional execution time for instant payments. The time can be formatted like HHMMSS or HH:MM.SS or similar. As separators are all not numeric chars allowed.

Separators can be mixed.

If you don't pass a time the time in the structure ***CSVConvertStruct*** takes place. This value is optional too.

The rest of the entries are buffers for future expansions and have to be set to 0.

15) Here you can optionally state the divider for the individual fields within the CSV file. The following characters are allowed.

; Semicolon
, Comma
T The value of T is internally used as tab character.

If you don't deliver a value here, it will be used the semicolon internally (**;**).

16) Here you can state optionally, if a character and which character for the limitation of the fields should be used. Sometimes fields are included in quotation marks.

The following characters are allowed.

„ Quotation marks
' inverted comma
/ diagonal slash

Even some fields are only included in delimiters, you have to state the character here. Sometimes fields are enclosed in quotes.

As a standard the function emanates from the fact that no delimiters are available.

17) If you want to read in amounts with thousand-separators (e.g.: 3.123.654,49) you can state here optionally the character for the thousand-separator. If there is no value set, internally the full stop (**.**) is used for the thousand-separators.

18) Set here optionally the character for the decimal point. Allowed is the comma and the full stop (US-representation). If there is nothing set, internally the comma (German representation) is used.

19) Optional you can pass a number, how may lines (header lines) should be skipped. A negative number is equal as zero. A number greater than 255 is equal 255.

Please note:

In which order the fields are included in the CSV file is not relevant. The 50 first fields (max.) within the CSV file are analyzed. In this there have to be included the relevant data independent of the position.

35.4 Returncodes

0 Everything was successful.

-1 The input file (XML-file) couldn't be found.

-2 The path for the output file (the XML-file which has to be generated) is formal invalid (less than 2 characters).

-3 The declaration of the path (with file name) for the export file (XML-file) is too long (longer than 200 characters).

-4 The drive in the path for the output file is not ready.

-5 The directory which was named for the output file doesn't exist.

-6 The data carrier, which was named for the output file is write protected.

-7 The reading of the CSV-file was not successful.

-999 The API wasn't initialized yet. Please first call the function SepaTools_Init.

36 SepaTools_GetDTAProtokoll

36.1 Purpose of the function

With this function you are able to read out the error free converted data records and also the faulty data records and also log it. this is the same function which is also used when converting DTA-files.

The function has to be called multiple times until the returncode has the value of <>0. With the parameter “Fehler” you regulate if the valid or the faulty data records are emitted.

36.2 Call of the function

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

36.3 Parameters

Please have a look to the original description of this function in the part of the conversion of DTA files.

36.4 Detail-Error-codes

Please have a look to the original description of this function in the part of the conversion of DTA files.

36.5 Returncodes

0	Everything was successful. There are more data available. You have to call the function repeatedly (with parameter First=0).
-3	The temporary file can’t be opened for reading.
-4	You have reached the end of the data base. There are no more data available.
-999	The API wasn’t initialized yet. Please first call the function SepaTools_Init.

36.6 Example for CSV-Converting

With the following example file (credit transfer) the correct realization should be shown:

```
Kundennummer1;600,25;Meierhofer Rita;Gehalt;Bemerkung1;26580070;732502200
Kundennummer2;1800,50;Altmannhofer Hans;Rechnung;Bemerkung2;60050101;2777939
```

Note:

The field of “Kundennummer” and “Bemerkung” are not necessary for the conversion, however they are included in the example.

Otherwise the file includes the amount, the name of the recipient, the reason of payment, bank code number and bank account number. Instead of bank code number and bank account number, you also can include the BIC and the IBAN.

The data of the initiator are taken from the structure.

36.6.1 Set of the structure

struct CSVConvertStruct

	Feld/Variable		Content
{	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. . . the following are set to 0]
	char	Trennzeichen[1+1]	
	char	Grenze[1+1]	
	char	Tausend[1+1]	
	char	Komma[1+1]	
	char	Reserve[200]	
}			

37 To Create DTAZV files

The API can create DTAZV-files for the German foreign payments. Optional there can be created XML-files instead of the DTAZV-files, if the dataset is “SEPA-valid”.

Please see also the documentation of “Deutsche Kreditwirtschaft” (DK) which is posted on the website www.sepa-tools.de, under capital “Sonstige Infos”.

To create DTAZV -files, there are required three functions, which are shown below.

- Call of SepaTools_CreateAZV	Initializing the process.
- Call of SepaTools_WriteAZV	This call can be repeated nearly unlimited. The maximum value depends only from the memory of your computer. In every call, the structure with the data of the payment will be passed.
…	
- Call of SepaTools_CloseAZV	This application will be closed and the file will be written to the disk.

Inside of the functions, many plausibility checks will appear.

38 SepaTools_CreateAZV

38.1 Purpose of the function

With this function, the creation of the DTAZV-file will be initialized. Further the data of the presenter (submitter) will be passed here. The values will be passed in a structure, which is passed as parameter.

38.2 Function call

int SepaTools_CreateAZV(AZVCreateStruct *CreateStruct)

38.3 Parameters

CreateStruct Please pass here the structure CreateStruct. The structure is shown below. All strings are null-terminated. The length of the char array is always one char longer as the real string.

All strings will be internally checked, if it is a valid charset for foreign payments. Is this not the case, then invalid chars will be deleted. Lower chars will be converted to upper chars.

Already in this function will be checked, if the output path is writeable.

struct AZVCreateStruct

{	char	ExportPfad[255+1]	Drive and path for the export of the file to create.
	char	AZVName[35+1]	The filename of the DTAZV-file to create [1].
	char	EinreicherName1[35+1]	The name of the presenter of the payment. (Q5)
	char	EinreicherName2[35+1]	Add. name of the presenter of the payments (optional) (Q5)
	char	EinreicherStrasse[35+1]	Address of the presenter of the payments (optional) (Q5)
	char	EinreicherOrt[35+1]	City of the presenter of the payments (optional) (Q5)
	char	EinreicherKonto[10+1]	German account or cutomer number (Q4)
	char	EinreicherBLZ[8+1]	Bank identification code of the receiving bank (Q3)
	char	AusfDatum[8+1]	Execution date in the form DDMMYYYY (Q8)
	int	TrySEPA	Optionally check because SEPA-ability [2]
	char	MsgId[35+1]	Optionally Message-Id for the optional XML-file [3]
	char	PmInfo[10+1]	Optional a short cut for the building a Payment-Info-Id [4]
	char	EndToEndId[10+1]	Optional a short cut for the building an End-To-End-Id [5]
	char	Reserve[500]	Reserved for later use
}			

Additional notes:

1) If an empty string is passed, the name of the AZV-file will be set to the name “DTAZV” (without extension).

2) If you pass the value “1” in this field, there will be a check if the dataset is SEPA-valid. In this case, the record will be written in a SEPA XML-file and not in then DTAZV-file.

   The name of the XML-file is set to the name of the DTAZV-file with the extension “xml”. The same export path as for the DTAZV-file is used.

3) The message-id should be a unique identification of the XML-file. This is only necessary, if the variable “TrySEPA” is set to the value “1”.

   If an empty string is passed, then internally will be generated automatically a Message-Id (as shortcut then Value “AZVtoXML ist used). If a string until 10 chars is passed, then this chars will used as shortcut for the building of the Message-Id.

4) If the value of the variable “TrySEPA” is set to “1”, then you can pass a shortcut (with max. 10 chars) for the building of the Payment-Info-Id. If an empty string is passed, the shortcut “AZVtoXML” is used.

5) If the value of the variable “TrySEPA” is set to “1”, then you can pass a shortcut (with max. 10 chars) for the building of the End-to-End-Id. If an empty string is passed, the shortcut “AZVtoXML” is used.

38.4 Return codes

0	Everything was successful.
-1	The name of the export path was too short (less than 2 characters).
-2	The named directory for the export file does not exist.
-3	The disk, which was named for the path of the export file is write protected.
-6	The name of the export path is too long (longer than 200 characters).
-7	The named drive in the path of the export file is not ready (possibly no disk inserted).
-8	The name of the XML-file is too short (less than 3 characters).
-9	The function SepaTools_CreateAZV was already called. Before it will be called again, then function SepaTools_CloseAZV must be called. The functions to create a DTAZV-file are not “multi thread” able.
-10	The database (sepa.dat, sepa.idx) is not actually. You need the newest database, which contains country- and currency information.
-11	No Name for the presenter (submitter) was passed.
-12	The bank identifier code of the file receiving bank is invalid.
-13	The given execution date is invalid. It must be in the range from “today” until 15 days later.
-999	The API hasn’t been initialized. Please first call the function SepaTools_Init.

39 SepaTools_WriteAZV

39.1 Purpose of the function

With every call of this function, exact one record of the payment is passed. Many plausibility checks may appear. If there are no plausibility errors, record will be written in the DTAZV file.

If the variable TrySEPA in the CreateAZVStruct is set to the value “1”, then every positive checked data record will additionally checked if it is SEPA valid. Is this, then the record will be written to the XML file instead of the DTAZV file.

39.2 Function call

int SepaTools_WriteAZV(AZVWriteStruct *WriteStruct)

39.3 Parameters

WriteStruct A Pointer to the structure WriteStruct will be passed as parameter. The structure is shown below. The length of the char array is always one char longer as the real string.

All strings will be internally checked, if it is a valid charset for foreign payments. Is this not the case, then invalid chars will be deleted. Lower chars will be converted to upper chars.

struct AZVWriteStruct

{	char	AuftragBLZ[8+1]	Bank identifier code of the client (T3)
	char	AuftragKonto[10+1]	Account number of the client (T4b)
	char	AuftragWhg[3+1]	ISO3-Code of the currency of the client (T4a)
	char	AusfTermin[8+1]	Execution date of the payment, form DDMMYYYY (T5) [1)]
	char	GebuehrBLZ[8+1]	Optional bank identifier code for fees (T6)
	char	GebuehrKonto[10+1]	Optional account number for fees (T7b)
	char	GebuehrWhg[3+1]	Optional currency for fees T17a)
	char	EmpfBIC[11+1]	BIC of the bank of the payee (T8) [2]
	char	EmpfBankLand[2+1]	ISO2-Code of the country of the payee bank (T9a) [3)]
	char	EmpfBankName1[35+1]	Name of the payee bank (T9b) [3)]
	char	EmpfBankName2[35+1]	Optional an additional of the payee bank (T9b)
	char	EmpfBankStrasse[35+1]	Optional an address of the payee bank (T9b)
	char	EmpfBankOrt[35+1]	Optional the city of the payee bank (T9b)
	char	EmpfLand[2+1]	ISO2-Code of the country of the payee (T10a)
	char	EmpfName1[35+1]	Name of the payee (T10b)
	char	EmpfName2[35+1]	Optional an additional name of the payee (T10b)
	char	EmpfStrasse[35+1]	Optional an Address of the payee (T10b)
	char	EmpfOrt[35+1]	Optional the city of the payee (T10b)
	char	Order1[35+1]	Order 1 for check payments (T11)
	char	Order2[35+1]	Order 2 for check payments (T11)
	char	IBAN[35+1]	IBAN Account number of the payee (T12) [4)]
	char	Whg[3+1]	ISO3-Code of the currency of the payment (T13)
	char	Betrag[15+1]	The ammount of the payment in „cent“ (T14) [5)]
	char	Zweck1[35+1]	Purpose Line 1 of the payment (T15)
	char	Zweck2[35+1]	Purpose Line 2 of the payment (T15)
	char	Zweck3[35+1]	Purpose Line 3 of the payment (T15)

char	Zweck4[35+1]	Purpose Line 4 of the payment (T15)
char	Weisung1[2+1]	Instruction 1, see DK-documentation (T16)
char	Weisung2[2+1]	Instruction 2, see DK-documentation (T17)
char	Weisung3[2+1]	Instruction 3, see DK-documentation (T18)
char	Weisung4[2+1]	Instruction 4, see DK-documentation (T19)
char	Zusatz[25+1]	Optional an additional information (T20)
char	Entgelte[2+1]	Code for fee (T21) [6)]
char	ZVArt[2+1]	Code for payment kind, see DK-documentation (T22)
char	FreiText[27+1]	Optional an additional text for the client (T23)
char	Ansprechpartner[35+1]	Optional an contact or phone number for the client (T24)
char	Reserve[500]	Reserved for later use

}

Additional notes:

1) The execution date must be in the range from “today” and the following 15 days. Is no execution date passed, the execution date is set to the date which is passed to the structure AZVCreateStruct.

2) If the bank of the payee is a German bank then it can be passed a German bank identifier code. The required three slashes (///) will internally added automatically.

   For check payments, this field must not be used. Is it used, it will deleted automatically.

3) Is no value passed to the field “BIC”, this field is mandatory. For check payments this filed must not be used. If used, it will be deleted (intern) automatically.

4) The leading slash (see DK-documentation) will be set internal automatically. For check payments, this field must not be used. Is it used, it will deleted automatically.

5) The amount of the payment is always shown in the smallest unit. For example, the amount 2345.87 USD is shown as 234587.

6) Defines for the codes of fee, see DK-documentation.

   00 Fees are shared between the client and the payee
   01 All fees are charged to the client
   02 All fees are charged to the payee

39.4 Return codes

Note:

For return codes >=0 (no error), in the code will be stored additional information. This information is bitwise coded.

The bit coding for values >=0 have the following meaning.

0	Everything was successful.
1	The IBAN of the Payee was checked. It is possibility wrong. If this value is really an IBAN, you should check it.
2	The ISO-2 country code and the ISO-3 currency code was checked against. The currency does not fit to this country code. Please check this.
4	No purpose was passed.
8	For this data record, an entry to the optional SEPA XML-file was created.

The following codes are as negative integer values to see.

-2	The function SepaTools_CreateAZV was not called before this function.
-3	The DTAZV File could not be created.
-4	The data record (T-record) could not be written in the DTAZV file.
-101	The bank identifier code of the client is invalid.
-102	The account number of the client is invalid (check digit).
-103	The currency of the client is invalid.
-104	The execution date is invalid.
-105	The bank identifier code for the fee is invalid.
-106	The account number for the fee is invalid.
-107	The currency for the fee is invalid.
-108	IBAN or account number of the payee must be passed.
-109	For the chosen payment kind, the country must be passed (because missing BIC).
-110	The country code of the payee is invalid.
-111	The country code for the bank of the payee is unknown.
-112	For the chosen payment kind, the name of the bank of the payee must bee passed (missing BIC).
-113	The name of the payee must be passed.
-114	The IBAN which was passed is invalid (urgent transfer).
-115	The currency of the payment is invalid.

-116	The amount is zero (“0”).
-117	No client account number was passed.
-121	The instruction key 1 is with this payment kind invalid.
-122	The instruction key 2 is with this payment kind invalid.
-123	The instruction key 3 is with this payment kind invalid.
-124	The instruction key 4 is with this payment kind invalid.
-129	The combination of the instruction keys is invalid (see DK-documentation).
-130	The code for the fees is with this payment kind invalid.
-131	The code for the payment kind is invalid (see DK-documentation).
-151	For the payment kind “11” must be passed a BIC for the payee bank.
-161	For then payment kind “11” the instruction key 1 muss be 10, 11, 12 (if exist).
-162	For then payment kind “11” the instruction key 2 muss be 10, 11, 12 (if exist).
-163	For then payment kind “11” the instruction key 3 muss be 10, 11, 12 (if exist).
-164	For then payment kind “11” the instruction key 4 muss be 10, 11, 12 (if exist).
-999	The API hasn’t been initialized. Please first call the function SepaTools_Init.

40 SepaTools_CloseAZV

40.1 Purpose of the function

This function call closes the DTAZV file. According checksums will be written in the structure which is passed.

40.2 Function call

int SepaTools_CloseAZV(AZVCloseStruct *CloseStruct)

40.3 Parameters

CloseStruct A Pointer to the structure CloseStruct will be passed as parameter. The structure is shown below. The structure will be passed empty and filled from the API.

struct AZVCloseStruct

{	int	NumAZV	Number of error free created DTAZV records.
	int	NumSEPA	Number of SEPA records.
	int	NumInvalid	Number of invalid records.
	char	BetragAZV[12+1]	Sum of the DTAZV-amounts in “cent”.
	char	BetragSEPA [12+1]	Sum of then SEPA-amounts in “cent”.
	char	BetragInvalid[12+1]	Sum of the amounts of invalid records.
	char	Reserve[100]	Reserved for later use.
}			

40.4 Return codes

0	Everything was successful.
-2	The function SepaTools_CreateAZV was not called.
-4	The data record (Z-record) could not be written in the DTAZV file.
-999	The API hasn’t been initialized. Please first call the function SepaTools_Init.

41 SepaTools_GetLandWhg

41.1 Purpose of the function

This function enabled plausibility forward checks of country codes and currency codes. This function is also used intern plausibility checks.

41.2 Function call

int SepaTools_GetLandWhg(const char *Land,
const char *Whg,
LandWhgStruct *LandWhg)

41.3 Parameters

Land Please pass in this parameter the ISO-2 country code (i.e. DE, US, AT etc.) of the desired country, which should be checked. If you only would check the currency, you can pass here an empty string.

Whg Please pass in this parameter the ISO-3 currency code (i.e. USD, EUR etc.) of the desired currency, which should be checked. If you only would check the country, you can pass here an empty string.

LandWhg A Pointer to the structure LandWhgStruct, in which the results will be returned.

The structure is shown below.

struct LandWhgStruct

{	int	Info	Information about the result of the function call [1)]
	char	LandISO2[2+1]	ISO2-Code of the founded country.
	char	LandISO3[3+1]	ISO3-Code of the founded country.
	int	LandNum	Numerical country code.
	char	LandName[40+1]	Name of the country.
	char	WhgISO3[3+1]	ISO3-Code of the founded currency.
	char	WhgName[40+1]	Name of the currency.
	char	Reserve[100]	Reserved for later use.
}			

Additional note:

1 The information in this field (Info) are bitwise stored. The single bits have the following meaning:

0 No special notes.
1 The country code which is passed is invalid or was not found.
2 The currency code which is passed is invalid or was not found.
4 Currency code and country code does not fit.

41.4 Return codes

0	Everything was successful.
-1	The database could not be opened.
-10	The database is not actually. You need the actual database, which contains the country- and currency-information.
-999	The API hasn’t been initialized. Please first call the function SepaTools_Init.

42 SepaTools_SetVersionUndLand

42.1 Purpose of the function

With this function you can determine the XML-version and the country or the presenter for the XML-file, Germany, Austria and Switzerland are available. The structure of the file has insignificant differences.

42.2 Function call

int SepaTools_SetVersionUndLand(int Version, intLand)

42.3 Parameters

Version See following table.

Legend fort he table:

Gültig ab	Date oft he publishing
Version DK	Version Deutsche Kreditwirtschaft
Version RB	Rule Book Österreich
Version ST	Version number for SepaTools, which is to pass to the function SetVersionUndLand
Land	Country for which the version is valid
Formate	Formats for credit transfer and direct debit

Gültig ab	*Version*	*Land*	*Formats*	*Remarks*
22.11.2009	DK 2.4 RB 3.2 Ver. ST=1	DE AT	pain.001.002.02 pain.008.002.01	No comments
20.11.2010	DK 2.5 RB 5 Ver. ST=2	DE AT	pain.001.002.03 pain.008.002.02	No comments
17.11.2012	DK 2.7 RB 6 Ver. ST=3	DE AT	pain.001.003.03 pain.008.002.02	There is no version 2.6, because this is technical ident with the version 2.5.
17.11.2012	DK 2.7 RB 6 Ver ST=4	AT	pain.001.003.03 pain.008.002.02	Creates a XML-file according to rule-book 6.0 for the country Austria. It is only valid in combination with the value Land_AT=1. If you use this entry common with the value Land_DE, then the value will automatically set to **3**.
20.11.2016	DK 3.0 RB 7 bis 9 Ver. ST=5	DE AT	pain.001.001.03 pain.008.001.02	Creates a XML-file according to DK-version 3.0 and Austrian Rulebook 7.0 to 9.0. Version 3.0 has to be provided by the banks from November 2016
19.11.2017	DK 3.1 RB 7 bis 9 Ver. ST=6	DE AT	pain.001.001.03 pain.008.001.02	Creates a XML-file according to DK-version 3.1. Version 3.1 has to be provided by the

Gültig ab	*Version*	*Land*	*Formats*	*Remarks*
				banks from November 2017
17.11.2019	DK 3.3 RB 7 bis 9 Ver. ST=7	DE AT	pain.001.001.03 pain.008.001.02	Creates a XML-file according to DK-version 3.3. Version 3.3 has to be provided by the banks from November 2019 There are only clarifications to the INST Payments. In all other cases, the version is identically to the version 3.1.
19.11.2023	DK 3.7 RB open Ver. ST=8	DE AT	pain.001.001.03 pain.008.001.02 pain.001.001.09 pain.001.001.09.AXZ	This version pain.001.001.09 is the future format for credit transfer. When the banks support it, it can be used. With this format you also can create Instant payments with the optional possibility of time for the execution date. Outside from that you can create with the format pain.001.001.09 cross border payments (outside of SEPA room). This is mandatary from November 2025. The DTAZV format is then obsolet.
Nov. 2011	??? Ver. ST=50	CH	pain.001.01.003 pain.008.001.02.ch.02 pain.008.001.02.chsdd.02	This value is used for the API in Switzerland. Although there are actually no different versions in Switzerland, this value is used for compatibility reasons.
Nov 2022	??? Ver. ST=51	CH	pain.001.01.009 pain.008.001.02.ch.03 pain.008.001.02.chsdd.02	Future Version.

Land You can define the country or the presenter. The following values are possible:

Land_DE = 0 Germany

Land_AT = 1 Austria

Land_CH = 2 Switzerland

Note:

If you don’t call this function, the norm is ZKA-Version 2.7 for Germany.

42.4 Return codes

0 Everything was successful.

-1 There was delivered an invalid version.

-2	There was delivered an invalid country code.
-999	The API hasn’t been initialized. Please first call the function SepaTools_Init.

43 SepaTools_XMLLesenInit

43.1 Purpose of the function

With this function the reading of the SEPA XML-file is initialized. There are extensive plausibility checks. After this, the XML-file is converted in a temporary file. These can be read out with the following calls of SepaTools_XMLLesen.

43.2 Function call

int SepaTools_XMLLesenInit(const char *Pfad, const char *FN)

43.3 Parameters

Pfad	A pointer is passed to the path, in which the XML-file is located. The path must be null-terminated.
FN	A pointer is delivered to the file name of the XML-file. The path must be null-terminated.

43.4 Return codes

0	Everything was successful.
-1	There was delivered a formal invalid path (less than 2 characters).
-2	The path, which is stated in the directory doesn't exist.
-3	The XML-file wasn't found.
-5	The temporary file couldn't be generated. Please check the path which was delivered by the function SepaTools_Init for the temporary file (TempPath).
-6	The delivered path is too long (>200 characters).
-7	The directory which is stated in the path is not ready
-8	The name of the XML-file is too short <3 characters.
-11	The XML-file could not be read. The file could be too large for the available main memory.
-12	The XML-file is no valid SEPA XML-file. Please proof the XML file according to the SEPA Rulebook.
-13	The Group-Information can't be written in the temporary file.
-14	In the XML-file couldn't be found any SEPA payment orders.
-15	The payment information couldn't be written in the temporary file.
-999	The API wasn't initialized. Please first call the function SepaTools_Init.

44 SepaTools_XMLLesen

44.1 Purpose of the function

With this function it is possible to read out the several data records, if there was a previous call of the function SepaTools_XMLLesenInit. Please set the value First to 1 when you call the function for the first time, else value=0.

At the moment, when all data records are read out, the temporary file is automatically closed and deleted.

If you don't call the function until the reading of the temporary fields is completed, you have to call the function SepaTools_XMLLesenClose, to delete the temporary file.

44.2 Function call

int SepaTools_XMLLesen(const int *First, XMLGroupStruct *GroupStruct
XMLReadStruct *ReadStruct)

44.3 Parameters

First Please set this value to 1, when you call the function for the first time. Because of this, the initialization values are set.

Please set the value to 0, before you call the function again.

GroupStruct This structure is only filled, at the moment when you call the function for the first time. The content stays there, expect the application is deleting the content.

The structure is shown below.

struct XMLGroupStruct

```
{   int     Direct debit          value=1, if it is a direct debit, else value=0.
    int     SummeAnzahl           Number of data records in the XML-file.
    chare   SummeBetrag[12+1]     Total sum of the amounts in the XML-file in cent.
    char    MsgId[35+1]           Message-Id of the XML-file.
    char    ErstDatum[8+1]        Date of execution of the XML-file (format DDMMYYYY).
    char    ErstZeit[8+1]         Time of execution of the XML-file (format HH:MM.SS).
    char    EinreicherName[70+1]  Name of the presenter of the XML-file)
}
```

RadStruct This structure is filled in with the data of the payment order every time when the function is called.

The structure is shown below.

struct XMLReadStruct

```
{   char    PmInfoId[35+1]        PaymentInfoId – Code of the order.
    char    AusfDatum[8+1]        Date of execution of the payment (format:DDMMYYYY).
```

char	AuftragName[70+1]	Name of the initiator
char	AuftragBIC[11+1]	BIC of the initiator
char	AuftragIBAN[35+1]	IBAN of the initiator
char	AuftragAbwName[70+1]	Differing name of the initiator (optional).
char	AuftragCI[35+1]	CI of the initiator (concerning direct debits)
char	EmpfName[70+1]	Name of the recipient of the payment.
char	EmpfBIC[11+1]	BIC of the recipient.
char	EmpfIBAN[35+1]	IBAN of the recipient.
char	EmpfAbwName[70+1]	Differing name of the recipient (optional).
char	Purpose[4+1]	Purpose of the payment (optional)
char	Betrag[12+1]	Amount in Cent.
char	EndToEndId[35+1]	Code of the single payment.
char	MandatId[35+1]	Code for the mandate only for direct debits
char	MandatDatum[8+1]	Date of the mandate (format DDMMYYYY).
char	SequenceType[4+1]	Sequence of the direct debit.
int	B2B	B2B-direct debit (<>0) or normal Direct debit (=0).
int	SammlerAnzahl	Number of payment orders in the collector
char	SammlerSumme[12+1]	Sum of the amounts of the collector in Cent.
char	Zweck1[70+1]	Purpose of the payment row 1
char	Zweck2[70+1]	Purpose of the payment row 2
int	Hinweis[30]	Field with up to 30 note characters.

}

44.4 Return codes

0	Everything was successful.
-1	The temporary file could not be opened.
-2	The function SepaTools_XMLLesenInit wasn't called yet.
-3	You have reached the end of the temporary file. There are no more data files available.
-4	The reading of the GroupHeaders wasn't successful.
-5	While reading the payments order, an error appeared.

Notes:

While converting the XML-file note code characters are generated. These notes do not mean, that the process has to be stopped. They only show differences within the XML-File.

There is a 30 characters long field with integer-values for each data record within in the structure XMLReadStruct. If single values are allocated stated with values <>0, these can be evaluated.

If it is a note concerning the GroupHeader of the XML-File, the note is available in each data record (the GroupHeader is unique).

If the notes concern the initiator data, these notes are available in each data record of the collector.

If the SEPA-XML data record is valid, all notes should be stated with 0.

The note code characters have the following meanings:

-301	There wasn't found a name for the presenter of the payment.
-302	There were presented address data in the GroupHeader, but no country code.
-303	There were found no data within the GroupHeaders.
-304	The XML-File includes no GroupHeader.
-401	In the collectors data no name for the client of the payment was found.
-402	There were presented address data for the client, but no country code.
-403	There couldn't be found any clients data.
-404	For the initiator of the collector, no BIC was found.
-501	There weren't found no recipients data (payment orders).
-502	There wasn't found a EndToEnd-Id for the payment order.
-503	There wasn't found any amount for the payments order.
-504	There wasn't found a name for the recipient of the payments order.
-505	There were presented address data for the recipient, but no country code.
-506	There wasn't appointed an IBAN for the payment recipient.

45 SepaTools_XMLLesenClose

45.1 Purpose of the function

This function closes and deletes the temporary file.

Did the function SepaTools_XMLLesen end with the return code -3 (End of the File), you don't have to call the function. A repeatedly call is not corruptive.

45.2 Function call

void SepaTools_XMLLesenClose()

45.3 Parameters

none.

45.4 Return codes

none.

46 SepaTools_GetBIC

46.1 Purpose of the function

This function can establish the BIC out of the German bank routing number.

Note:

The returned BIC correspond the BIC which is deposited at the Deutschen Bundesbank.

By converting of BLZ and bank account number into BIC and IBAN with the function SepaTools_ConvertBLZKonto the special factors, which were named by the credit institution are included.

The returned BIC by using this function can differ and can't be used for SEPA payment

46.2 Function call

int SepaTools_GetBIC(const char *BLZ, char *BIC)

46.3 Parameters

BLZ	Please pass a pointer to the bank routing number, for which the BIC should be found. The bank routing number has to be delivered null-terminated 8 characters long strings for Germany and 5 characters long strings for Austria.
BIC	Please pass a pointer to a memory area, where the founded BIC can be saved. The memory area has to be able to save 11+1 characters.

46.4 Return codes

0	Everything was successful. The BIC was found.
-1	The BIC couldn't be established because the BLZ was not found.
-2	The BLZ was found, even for this BLZ was no BIC available.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

47 SepaTools_CheckIBAN

47.1 Purpose of the function

By this function the IBAN can be proofed if it is right. Invalid characters will be removed bevor checking the IBAN. Lowercase characters will be converted to uppercased characters.

47.2 Function call

int SepaTools_CheckIBAN(const char *IBAN)

47.3 Parameters

IBAN	Deliver a pointer to the IBAN which has to be proofed.

47.4 Return codes

0	Everything was successful. The IBAN is valid.
-1	The database Sepa.dat couldn't be opened.
-2	There wasn't delivered an IBAN (empty string).
-3	The country code in the IBAN is not a country code from a SEPA country. The IBAN can't be used for SEPA.
-4	The length of the IBAN has not the country defined length.
-5	The test digit of the IBAN (Character 3 und 4) may only have numbers.
-6	The test digit of the IBAN is not valid. The IBAN can't be used for SEPA.
-7	This error result can only occur with German IBANs. The IBAN is formally correct. But the bank code, which was used to build the IBAN, was not found in the directory of the German bank codes.
-8	This error result can only occur with German IBANs. The IBAN is formally correct. But the check digit of the account number, which was used to build the IBAN, is not correct. The account number is invalid.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

48 SepaTools_CheckIBAN_Strong

48.1 Purpose of the function

By this function the IBAN can be proofed if it is right. In opposite to the function Sepa-tools_CheckIBAN , this function doesn't remove or convert any invalid characters. If any invalid ist find, then the function returns a negative return code.

48.2 Function call

int SepaTools_CheckIBAN(const char *IBAN)

48.3 Parameters

IBAN	Deliver a pointer to the IBAN which has to be proofed.

48.4 Return codes

0	Everything was successful. The IBAN is valid.
-1	The database Sepa.dat couldn't be opened.
-2	There wasn't delivered an IBAN (empty string).
-3	The country code in the IBAN is not a country code from a SEPA country. The IBAN can't be used for SEPA.
-4	The length of the IBAN has not the country defined length.
-5	The test digit of the IBAN (Character 3 und 4) may only have numbers.
-6	The test digit of the IBAN is not valid. The IBAN can't be used for SEPA.
-7	The IBAN contains invalid chars.
-8	This error result can only occur with German IBANs. The IBAN is formally correct. But the bank code, which was used to build the IBAN, was not found in the directory of the German bank codes.
-9	This error result can only occur with German IBANs. The IBAN is formally correct. But the check digit of the account number, which was used to build the IBAN, is not correct. The account number is invalid.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

49 SepaTools_GetBLZKonto

49.1 Purpose of the function

This function determines the bank code number, the bank account number, the BIC and the name of the bank out of the delivered IBAN. The function can be used for German, Austrian, Netherlands, Poland, Switzerland and Liechtenstein IBANs.

49.2 Call of the function

int SepaTools_GetBLZKonto(const char *IBAN, BLZKontoStruct *BLZKonto)

49.3 Parameters

IBAN Deliver here a pointer to the IBAN from which the data should be generated.

BLZKonto In this structure the delivered data are returned. The structure is shown in the following.

```
struct   BLZKontoStruct

{   char    Land[2+1]          Country identifier DE, AT, NL, PL, CH or LI
    char    BLZ[8+1]           The returned bank code number
    char    Konto[11+1]        The returned bank account number
    char    BIC[11+1]          The determined BIC
    char    BankName[50+1]     The name of the bank (out of SCL-Directory)
    char    Ort[30+1]          The location of the bank (out of BLZ-Directory)
    char    KontoLang[20+1]    An additional long account number 1)
    char    Reserve[148]       Buffer for later use.
}
```

Additional notes:

1) In Switzerland and Liechtenstein are longer account numbers than 11 characters used. To avoid the change of the origin field “Konto”, an additional field is inserted.

   Now in the field “Konto”, the shorted account number (down to 11 characters) is provided. In the additional field “KontoLang”, the full length account number is provided.

49.4 Returncodes

0	Everything was successful. The IBAN is valid. The conversion is done.

Attention!

The following three positive values are bit coded. It will be returned as one number. The bit codes values are shown in red color. These three bit coded values are only for information. They are only supplied if the operation is finished successful.

1	Everything was successful. The IBAN is valid. The conversion is done. There could be determined a BIC. But this BIC is not achievable by EBA.
2	The IBAN which was passed contains lower case chars. This lower case chars would be converted in uppercase chars.
4	The IBAN which was passed contains invalid chars. this invalid chars would be removed.
-1	The data base Sepa.dat could not be opened.
-2	There was no IBAN delivered (empty string).
-3	The country identifier of the IBAN corresponds to a country which has no IBAN. The IBAN cannot be used.
-4	The length of the IBAN doesn't correspond to the necessary length of the stated country.
-5	The test digit of the IBAN (character 3 and 4) may only consist out of numbers.
-6	The test digit of the IBAN is invalid. The IBAN can't be used for SEPA.
-7	The country identifier of the delivered IBAN is not „DE", „AT", CH and not "LI".
-8	There couldn't be determined a BIC. Probably the bank code number is invalid.
-11	The database BLZ.dat couldn't be opened. Please proof the path which is delivered in the function SepaTools_Init.
-12	The bank identifier code in the German bank directory was not found.
-13	The check digit of the account number of a German IBAN is invalid.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

50 SepaTools_CheckIBANLand

50.1 Purpose of the function

This function allows you to see, if a country has a IBAN/if the country participates in SEPA.

50.2 Function call

int SepaTools_CheckIBANLand(const char *Land, const int SEPA)

50.3 Parameters

Land Deliver a pointer to the 2 characters long ISO-Code (e.g. DE or IT) of the country, which you want to proof.

SEPA You can note here, if the proofing concerns only countries which have an IBAN and participate in SEPA or if it concerns all countries which have an IBAN.

The following values can be delivered.

0 The search runs in the chart with all countries, which have an IBAN, irrespective of their participation on SEPA.

1 (≠ 0) The search runs in the chart with all countries, which have an IBAN and participate in SEPA.

50.4 Return codes

0	Everything was successful. The IBAN is valid.
-1	The data base Sepa.dat couldn’t be opened.
-2	There wasn’t delivered a country (empty string).
-3	The country code wasn’t found in the charts. This concerns ether countries which have an IBAN irrespective of their participation in SEPA (Parameter SEPA = 0) or countries which have an IBAN and participate in SEPA (Parameter SEPA = 1).
-999	The API wasn’t initialized yet. Please first call the function SepaTools_Init.

51 SepaTools_CheckCI

51.1 Purpose of the function

This function can proof a Creditor Identification (CI) for its accuracy.

51.2 Function call

int SepaTools_CheckCI(const char *CI)

51.3 Parameters

CI	Deliver here a pointer to the CI, which has to be proofed.

51.4 Return codes

0	Everything was successful. The CI is valid.
-1	The data base Sepa.dat could not be opened.
-2	There was no CI delivered (empty string).
-3	The country code of the CI corresponds to a country which doesn't participate in SEPA. The CI can't be used for SEPA.
-4	The test digit of the CI (Character 3 and 4) may only be numbered.
-5	The test digit of the CI is invalid. The CI can't be used for SEPA.
-6	The Length of the passed CI is invalid. The length of the CI will be additionally checked, if it is possible. The check based on an officially document of the EPC. Not CIs from all country can be checked, because there has variable length.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

51.5 Known Length of CIs

The CIs of the following countries can be checked on the basis of the EPC information.

ID	*County*	*Length CI*
AT	Österreich	18
BE	Belgien	17 or 20
CH	Schweiz	18
CY	Zypern	11
CZ	Tschechien	12
DE	Deutschland	18
DK	Dänemark	15
EE	Estland	20

ID	County	Length CI
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

52 SepaTools_CheckESR

52.1 Purpose of the function

This function can proof an ESR number, which are often used in Switzerland. Only actual reference numbers with a length of 27 characters can be proofed.

Older versions are not supported.

52.2 Function call

int SepaTools_CheckESR(const char *ESR)

52.3 Parameters

ESR	Please pass a pointer to the ESR number which should be proofed.

52.4 Return codes

0	Everything was successful. The ESR number is valid.
-1	The ESR number (Reference) is not valid.
-2	The length of the ESR number is invalid (unequal 27)
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

53 SepaTools_GetESRPrZiff

53.1 Purpose of the function

This function calculates or recalculates the check digit of a Switzerland ESR number. If you pass an ESR number with 26 characters then the check digit will be added. If you pass an ESR number with 27 characters then the check digit will be replaced with the calculated check digit.

Only for actual reference numbers with a length of 27 characters the check digit can be calculated.

Older versions are not supported.

53.2 Function call

int SepaTools_GetESRPrZiff(char *ESR)

53.3 Parameters

ESR	Please pass a pointer to the ESR number for which the check digit should be calculated.

53.4 Return codes

0	Everything was successful. The ESR number is valid.
-2	The length of the ESR number is invalid (unequal 27 and unequal 26).
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

54 SepaTools_CheckSEPATeilnahme

54.1 Purpose of the function

This function can establish, on which SEPA process the bank participates.

54.2 Function call

int SepaTools_CheckSEPATeilnahme(const char *BIC, int *Info)

54.3 Parameters

BIC Deliver a pointer to the BIC, for which the process should be established. The BIC has ether 8 characters or 11 characters.

Info Deliver a pointer to an integer value in which the information should be saved. The information is saved by bits (or-interconnection) and has the following meaning:

1 The bank participates on the Credit transfers process (CT).

2 The bank participates on the normal direct debit process (DD) (Core-Direct debit.)

4 The bank participates on B2B-Direct debit.

8 The bank participates on the COR1-Direct debit (shorter deadline). Please note, COR1-Direct debit is usually not available cross border.

16 Reserved.

32 The bank participates on the Instant process (SCT Inst).

The values are connected by bits. If a bank participates in all of the four processes, the value of 15 is returned. Participates the bank only on the first two processes, the value of 3 is returned.

54.4 Return codes

0	Everything was successful. The data were read out of the database.
-1	For the stated BIC no data were found.
-2	There was delivered a BIC with a invalid length (<>8 und <>11).
-3	The delivered BIC exists, but it doesn't participate in SEPA
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

55 SepaTools_ConvertBLZKonto

55.1 Purpose of the function

With this function there is established a BIC and a IBAN out of a BLZ and bank account number. Not all Credit institutions published their calculation rules. This is the reason, why here can appear errors.

If special BICs and IBANs were deposited with corresponding BLZ and bank account number, this translation chart is involved in the establishment of BIC and IBAN out of bank identifier code an bank account number.

This function can in most instances also handle Austrian bank connections. Based on the delivered bank identifier code (5 characters for Austria and 8 characters for Germany), the API decides, if the stock of bank routing number of the Deutschen Bundesbank or those of the Österreichischen Nationalbank is used.

Note:

There is no proofing of the test digit concerning Austrian banks.

By converting of BLZ and bank account number into BIC and IBAN with the function SepaTools_ConvertBLZKonto the special factors, which were named by the credit institution are included.

It is possible that the BIC which is returned by this function differs to the BIC which is returned by the function SepaTools_GetBIC which has to be used for SEPA payments

55.2 Function Call

int SepaTools_ConvertBLZKonto(XMLConvertStruct *ConvertStruct)

55.3 Parameters

ConvertStruct In this structure the values are delivered and the results are returned.

While converting it is possible that a BLZ is replaced or a bank account number is completed. In this case the modified origin values are returned.

```
struct    XMLConvertStruct

{   char    BLZ[8+1]          BLZ, which has to be used.
    char    Konto[11+1]       Bank account number, which has to be used.
    char    BIC[11+1]         The established BIC is returned.
    char    IBAN[35+1]        The established IBAN is returned.
}
```

55.4 Return codes

Notes:

Return codes >=0 (no errors):

In the return code are saved additional information. This information is shown bitwise.

The deposit with bites for the return code >= 0 means the following:

0	Everything was successful. The data were read out of the data base.
1	Everything was successful. The bank account number was modified, because of special rules of credit institutions. The modified bank account number is returned in the structure.
2	Everything was successful. The bank routing number was replaced with another bank routing number, because of special rules of credit institutions. The new bank routing number is returned in the structure.
4	The Bank identifier code is flagged to be deleted. There was no bank identifier code named, which will follow. It is unknown when the bank identifier code will be deleted. Because of this, the bank identifier code can be used further. It is a note that the bank routing number will be deleted at any time.
8	The Bank identifier code is flagged to be deleted. There was named a bank routing number, which will follow and which can be used for establish the IBAN. The new bank routing number can be used.
16	The conversion of BLZ und Bank account number in to IBAN is completed. There were used experienced data. The probability of the correct conversion is high, but not guaranteed. Please proof the result.
32	The calculated IBAN can be used. But it is not unique. We advise to contact the client. This positive return code only can appear if it is a bank account number with 7 characters of the Deutsche Bank.

From here on the error codes have to be evaluated as negative integer-values.

-2	The delivered bank identifier code has a length unequal 5 or 8.
-3	The delivered bank account number has only one character. The bank account number must have a value higher than 9.
-4	The delivered bank account number couldn’t be found in the stock of bank identifier codes of the Deutsche Bundesbank, or the Österreichischen Nationalbank.

-5	The delivered bank routing number will be deleted by the Deutschen Bundesbank and can't be used any more (not Austrian bank identifier codes). In the next publication of the stock of bank routing numbers, this code will not be included anymore.
-6	The test digit of the bank account number is wrong. The bank account number can't be used in conversion (not Austrian bank identifier codes).
-7	The calculation of the IBAN is not unique. The IBAN can't be used because of safety.
-8	For the delivered bank identifier code, no BIC was found.
-9	The database Sepa.dat could not be opened. Please proof the path, which was delivered in the function SepaTools_Init.
-10	The account holding bank has suspended this bank identifier code in connection with the bank account number from the IBAN-calculation. The IBAN calculation is not possible.
-11	The delivered bank routing number has characters which are invalid.
-12	The delivered bank account number includes invalid characters or is empty.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

56 SepaTools_SetErfahrung

56.1 Purpose of the function

This function isn't necessary anymore.

The function is implemented longer because of compatibility reasons. The call of this function doesn't have any results.

56.2 Function call

int SepaTools_SetErfahrung(int Flag)

56.3 Parameters

Flag	Please deliver an integer value as a dummy.

56.4 Return codes

0	Everything was successful. There is always delivered the value of 0.

57 SepaTools_GetBankInfo

57.1 Purpose of the function

With this function information about banks which participate in SEPA can be called.

57.2 Function call

int SepaTools_GetBankInfo(const char *BIC, XMLBankInfoStruct *BankInfoStruct)

57.3 Parameters

BIC Please pass the BIC, for which you want to access the information. The BIC has a length of 8 or 11 characters.

BankInfoStruct In this structure the results are returned.

struct XMLBankInfoStruct

{	int	Art	Process on which the bank participates [1)].
	char	BIC[11+1]	BIC of the bank.
	char	BankName[50+1]	Description of the bank.
	char	LandKurz[2+1]	Country code of the country of the bank, e.g. DE or AT.
	char	Land[20+1]	Description of the country
	char	Strasse[27+1]	Address of the bank (mostly not available).
	char	Ort[50+1]	Place of the bank (mostly not available).
}			

Additional notes:

1) Please pass an integer value in which the information is stored. The information is deposited by bites (or-connections) and has the following meaning:

   1 The bank participates on the credit transfers process (CT).

   2 The bank participates on the normal direct debit process (DD) (Core-Direct debit).

   4 The bank participates on the B2B-Direct debit.

   8 The bank participates on the COR1-Direct debit (shorter deadline). Please note, COR1-Direct debit is usually not available cross border.

   16 Reserved.

   32 The bank participates on the Instant process (SCT Inst).

   The values are connected by bits. If a bank participates on the first four processes, the value of 15 is returned. Participates the bank only on the first two processes, the value of 3 is returned.

57.4 Return codes

0 Everything was successful. The data were read out of the database.

1	Everything was successful. The data were read out of the database. The passed BIC was updated with XXX.
-1	There couldn't be established any information about the delivered BIC. Maybe the BIC is wrong or the bank doesn't participate on one of the SEPA-processes.
-2	The database Sepa.dat couldn't be opened. Please proof the path which is delivered in the function SepaTools_Init.
-3	The delivered BIC exists, but it doesn't participate in SEPA
-4	The database BLZ.dat couldn't be opened. Please proof the path which is delivered in the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

58 SepaTools_GetBICInfo

58.1 Purpose of the function

With this function, you can get information from the bank routing number of German or Austrian banks. You have to pass the requested BIC and a structure, where the results are passed.

58.2 Function call

int SepaTools_GetBICInfo(const char *BIC, XMLBLZStruct *BLZStruct)

58.3 Parameters

BIC Please pass the requested BIC.

BLZStruct In this structure the results are returned.

BLZStruct In this structure the results are returned.

struct XMLBLZStruct

{	char	BLZ[8+1]	Bank routing number
	char	NameKurz[27+1]	Short name of the bank.
	char	NameLang[35+1]	Long name of the bank.
	char	PLZ[5+1]	Postal code
	char	Ort[25+1]	Place of the bank
	int	EigeneBLZ	Value=1, if BLZ of the bank account holding bank, value=2 if agency
	char	Verfahren[2+1]	test digit process according to Deutschen Bundesbank
	int	DelHinweis	value=1, if BLZ is flagged for deleting, else 0
	char	ChangeKz	change indicator: A = new data record D = cancellation U = unchanged M = modified
	char	NachfolgeBLZ[8+1]	Following BLZ, if available
	char	BIC[11+1]	BIC corresponding to the BLZ [1)]
}			

Additional note:

1) The returned BIC is included in the stock of bank identifier codes of the Deutschen Bundesbank. The necessary BIC for the SEPA payment can differ from this BIC because it is also influenced of other factors.

   Always use (for the SEPA payment) those BIC, which was generated with the function SepaTools_ConvertBLZKonto. At some banks the BIC depends on a special bank account number circle.

58.4 Return codes

0	Everything was successful. The data were read out of the database.
-1	For the passed BIC no information was found.
-2	The database BLZ.dat couldn’t be opened. Please proof the path which is delivered in the function SepaTools_Init.
-3	The passes BIC doesn’t belongs neither to a German nor to an. Austrian bank
-999	The API wasn’t initialized yet. Please first call the function SepaTools_Init.

59 SepaTools_GetIBANLand

59.1 Purpose of the function

With this function you are able to call information about the countries, which have a IBAN. The function has to be called repeatedly, until a return code of <>0 is returned.

59.2 Function call

int SepaTools_GetIBANLand(int First, XMLIBANLandStruct *IBANLandStruct)

59.3 Parameters

First Please pass a value > 0, if you call the function for the first time. With this the function is initialized internally. Please specify the value of 0 when you call the other functions.

IBANLandStruct In this structure the results are returned.

struct XMLIBANLandStruct

{	char	LandKurz[2+1]	The short description of the country e.g. DE oder AT
	char	LandName[30+1]	The name of the country.
	char	IntLandName[20+1]	The international name of the country e.g. AUSTRIA.
	int	LaengeIBAN	The length of the IBAN of the country (Germany=22).
	char	IBAN[35+1]	An exemplary IBAN for this country.
	int	IsSepa	Value=1, if the country participates additionally in SEPA, else 0
}			

59.4 Return codes

0	Everything was successful. The data was read out of the database.
-1	You have reached the end of the database. There are no more data available.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

60 SepaTools_GetSEPABank

60.1 Purpose of the function

With this function you are able to call information about the countries, which participate in SEPA. The function has to be called repeatedly, until a return code of <>0 is returned.

Note:

Please note, here are returned very much data records (big data volume).

60.2 Function call

int SepaTools_GetSEPABank (int First, char *Land, char *Ort,
XMLBankInfoStruct *BankInfoStruct)

60.3 Parameters

First Please state a value > 0, if you call the function for the first time. With this the function is initialized internally. Please provide the value of 0 before you call the function next time.

Land Please state here the short code for the country (e.g. DE, AT a.s.o.) for this country in which you want to search for banks which participate in SEPA.

If you deliver an empty string here, in the list of all counties is searched for banks which participate in SEPA.

Ort Please state here the name of a place or a part of the name of a place (e.g. Frankf). If you have stated a short code for the country, there is searched for all banks in all places of the named country.

If you didn't state a short code for the country, the banks will be searched analog the search codes (places) independence of a country.

You can state an empty string here. As a result, all banks are searched.

BankInfoStruct In this structure the results are returned.

struct XMLBankInfoStruct

```
{   int     Art                 Process on which the bank participates. 1).
    char    BIC[11+1]           BIC of the bank.
    char    BankName[50+1]      Description of the bank.
    char    LandKurz[2+1]       Short code for the country of the bank e.g. DE oder AT.
    char    Land[20+1]          Description of the country.
    char    Strasse[27+1]       Address of the bank (often not available).
    char    Ort[50+1]           Place of the bank (often not available).
}
```

Additional notes:

1) An integer value is delivered. In this the information is saved. The information is deposited by bites (or-connections) and has the following meaning:

   1 The bank participates on the credit transfers process (CT).

   2 The bank participates on the normal direct debit process (DD) (Core-Direct debit).

   4 The bank participates on the B2B-Direct debit.

   8 The bank participates on the COR1 process.

   16 Reserved.

   32 The bank participates on the Instant process (SCT Inst).

   The values are connected by bits. If a bank participates in the first three processes, the value of 7 is returned. Participates the bank only on the first two processes, the value of 3 is returned.

60.4 Return codes

0	Everything was successful. The data were read out of the database.
-1	You reached the end of the database. There is no more information available.
-2	The database Sepa.dat couldn’t be opened. Please proof the path which is delivered in the function SepaTools_Init.
-3	As a short code for the country was stated a one characters long value. Only two characters long values (e.g. DE oder AT) or empty string is allowed.
-4	The database BLZ.dat couldn’t be opened. Please proof the path which is delivered in the function SepaTools_Init.
-999	The API wasn’t initialized yet. Please first call the function SepaTools_Init.

61 SepaTools_GetBLZ

61.1 Purpose of the function

With this function it is possible to call information out of the stock of bank routing number of the Deutschen Bundesbank or the Österreichischen Nationalbank. The function has to be called repeatedly until the return code is <>0.

61.2 Function Call

int SepaTools_GetBLZ(char *Land, int *First, char *Ort, XMLBLZStruct *BLZStruct)

61.3 Parameters

Land If you deliver AT for Austria a short code for the country, the bank routing number is searched in the stock of the bank identifier code of the Österreichischen Nationalbank.

In all other cases, even is an empty string is delivered it is searched in the stock of bank routing number of the Deutschen Bundesbank.

First Please state a value > 0, if you call the function for the first time. With this the function is initialized internally. After the first call of the function, the value is set to 0 by the API automatically.

Ort Please state the name of a place or a part of the name as a search item.

You also can state an empty string. As a result, all bank routing number are searched.

BLZStruct In this structure the results are returned.

struct XMLBLZStruct

{	char	BLZ[8+1]	Bank routing number
	char	NameKurz[27+1]	Short name of the bank.
	char	NameLang[35+1]	Long name of the bank.
	char	PLZ[5+1]	Postal code
	char	Ort[25+1]	Place of the bank
	int	EigeneBLZ	Value=1, if BLZ of the bank account holding bank, value=2 if agency
	char	Verfahren[2+1]	test digit process according to Deutschen Bundesbank
	int	DelHinweis	value=1, if BLZ is flagged for deleting, else 0
	char	ChangeKz	change indicator: A = new data record D = cancellation U = unchanged M = modified
	char	NachfolgeBLZ[8+1]	Following BLZ, if available
	char	BIC[11+1]	BIC corresponding to the BLZ [1)]
}			

Additional note:

1) The returned BIC is included in the stock of bank identifier codes of the Deutschen Bundesbank. The necessary BIC for the SEPA payment can differ from this BIC because it is also influenced of other factors.

   Always use (for the SEPA payment) those BIC, which was generated with the function SepaTools_ConvertBLZKonto. At some banks the BIC depends on a special bank account number circle.

61.4 Return codes

0	Everything was successful. The data were read out of the database.
-1	You have reached the end of the database. There are no more data available.
-2	The database BLZ.dat couldn't be opened. Please proof the path which is delivered in the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

62 SepaTools_FindBLZ

62.1 Purpose of the function

Information out of the stock of BLZ of the deutsche Bundesbank or the Österreichische Nationalbank concerning a delivered bank routing number is called by this function. The function has to be called repeatedly, until the return code of <>0 is retuned.

The retuned information is included in the structure XMLBLZStruct.

62.2 Function call

int SepaTools_FindBLZ(char *Land, const *BLZ, int *First, XMLBLZStruct *BLZStruct)

62.3 Parameters

Land If you deliver AT for Austria as a short code for the country, the bank routing number is searched in the stock of the bank routing number of the Österreichischen Nationalbank.

In all other cases, even there is passed an empty string it is searched in the stock of bank routing numbers of the Deutschen Bundesbank.

BLZ Deliver a pointer to the bank routing number, for which the information should be searched. Please call the function repeatedly, because it is possible that the BLZ is included multiple times in the stock of BLZ of the Deutschen Bundesbank

First Please state a value > 0, if you call the function for the first time. With this the function is initialized internally. After the first call of the function, the value is set to 0 by the API automatically.

BLZStruct In this structure the results are returned.

struct XMLBLZStruct

{	char	BLZ[8+1]	Bank routing number
	char	NameKurz[27+1]	Short name of the bank.
	char	NameLang[35+1]	Long name of the bank.
	char	PLZ[5+1]	Postal code
	char	Ort[25+1]	Place of the bank
	int	EigeneBLZ	Value=1, if BLZ of the bank account holding bank, value=2 if agency
	char	Verfahren[2+1]	test digit process according to Deutscher Bundesbank
	int	DelHinweis	value=1, if BLZ is prebooked for deleting, else 0
	char	ChangeKz	change indicator: A = new Data record D = cancellation U = unchanged M = modified
	char	NachfolgeBLZ[8+1]	Following BLZ, if available
	char	BIC[11+1]	BIC corresponding to the BLZ [1)]

}

Additional note:

1) The returned BIC is included in the stock of bank routing number of the Deutschen Bundesbank. The necessary BIC for the SEPA payment can differ from this BIC because it is also influenced of other factors.

   Always use (for the SEPA payment) those BIC, which was generated with the function SepaTools_ConvertBLZKonto. At some banks the BIC depends on a special bank account number circle.

62.4 Return codes

0	Everything was successful. The data were read out of the database.
-1	You have reached the end of the database. There are no more data available.
-2	The database BLZ.dat couldn't be opened. Please proof the path which is delivered in the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

63 SepaTools_SearchBLZ

63.1 Propose of the function

Information of the stock of BLZ of the Deutschen Bundesbank or the Österreichischen Nationalbank concerning a delivered bank routing number can be called by this function. The function has to be called repeatedly until the return code is <>0.

The returned information is included in the structure XMLBLZStructure.

It is possible that a bank routing number with less as 8 characters (5 in Austria) can be stated as a search term (in contrast to the function SepaTools_FindBLZ). The search is successful, when the characters, which are given as bank routing number with the first characters of the bank identifier code of the stock of the bank identifier codes accord.

If there is an empty string as a bank routing number delivered, all bank routing number of the country are listed (DE or AT).

63.2 Function call

int SepaTools_SearchBLZ(char *Land, const *BLZ, int *First, XMLBLZStruct *BLZStruct)

63.3 Parameters

BLZStruct In this structure the results are returned.

Land If you deliver AT for Austria as a short code for the country, the bank routing number is searched in the stock of the bank routing number of the Österreichischen Nationalbank.

In all other cases, even is an empty string is delivered it is searched in the stock of bank identifier codes of the Deutschen Bundesbank

BLZ Deliver a pointer to the bank routing number, for which the information should be searched. Please call the function repeatedly, because it is possible that the BLZ is included multiple times in the stock of BLZ of the Deutschen Bundesbank

First Please state a value > 0, if you call the function for the first time. With this the function is initialized internally. After the first call of the function, the value is set to 0 by the API automatically.

BLZStruct In this structure the results are returned.

struct XMLBLZStruct

```
{   char    BLZ[8+1]            Bank routing number
    char    NameKurz[27+1]      Short name of the bank.
    char    NameLang[35+1]      Long name of the bank.
    char    PLZ[5+1]            Postal code
    char    Ort[25+1]           Place of the bank
    int     EigeneBLZ           Value=1, if BLZ of the bank account holding bank
```

		Value=2 if agency
char	Verfahren[2+1]	Test digit process according to Deutscher Bundesbank
int	DelHinweis	Value=1, if BLZ is flagged for deleting, else 0
char	ChangeKz	change indicator: A = new Data record D = cancellation U = unchanged M = modified
char	NachfolgeBLZ[8+1]	Following BLZ, if available
char	BIC[11+1]	BIC corresponding to the BLZ [1)]

}

Additional note:

1) The returned BIC is included in the stock of bank routing number of the Deutschen Bundesbank. The necessary BIC for the SEPA payment can differ from this BIC because it is also influenced of other factors.

   Always use (for the SEPA payment) those BIC, which was generated with the function SepaTools_ConvertBLZKonto. At some banks the BIC depends on a special bank account number circle.

63.4 Return codes

0	Everything was successful. The data were read out of the database.
-1	You have reached the end of the database. There are no more data available.
-2	The database BLZ.dat couldn't be opened. Please proof the path which is delivered in the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

64 SepaTools_CheckPruefziffer

64.1 Purpose of the function

With this function you can proof if the bank account number of a German bank is valid.

64.2 Function call

int SepaTools_CheckPruefziffer(const char *BLZ, const char *Konto)

64.3 Parameters

BLZ	Please deliver an 8 characters long bank routing number of a German bank, from which a bank account number should be proofed
Konto	Please deliver the bank account number which should be proofed.

64.4 Return codes

0	Everything was successful. The bank account number is valid.
-1	The bank account number is invalid.
-2	The BLZ.dat couldn't been opened. Please proof the path which is delivered in the function SepaTools_Init.
-3	The delivered bank routing number couldn't be found in the stock of the BLZ-of the Deutschen Bundesbank.
-4	For Austrian banks no (bank routing number has 5 characters) test digit can be calculated.
-5	No account number, or the account number zero (0) was passed.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

65 SepaTools_GetPurposeCode

65.1 Purpose of the function

With this function, you can test a purpose code or a category purpose code if it exists. With an existing code, the structure returns additional information's of the code.

The available purpose codes are the same as in the ISO External Code Sets listed.

65.2 Function call

int SepaTools_GetPurposeCode(const char *Code, PurposeStruct *Purpose, int Flag)

65.3 Parameters

Code Please pass a pointer to the desired purpose code (i.e. BENE, SALA etc.). The code must be null terminated, 4 characters long.

Purpose Please pass a pointer to the structure ***PurposeStruct***. The structure is shown in then following.

Flag Please provide a Flag which means how a missing German translation is to interpret. The following values are valid:

0 By a missing translation, the value **N/A** is provided.
1 By a missing translation, the string "Keine Übersetzung" is displayed.
2 The German translation is provided with the original English term.

PurposeStruct In this structure the results of the function are provided..

struct PurposeStruct

{	int	Prio	Priority of the purpose code [1)]
	int	Typ	The type of the purpose code [2)]
	char	Code[4+1]	The 4 characters long purpose code
	char	Klasse[30+1]	The classification of the purpose code Codes
	char	Name[60+1]	The original term of the code
	char	Deutsch[60+1]	The German translation of the code (if available)
	char	Reserve[100]	Reserved for later use
}			

Additional notes:

1) With this code the "subjective" importance of the code is indicated. the following values are possible.

0 This purpose code is used frequently, i.e. BENE or SALA etc.
1 The other purpose codes contain the value 1.

2) Purpose codes are used in the transaction part as well the category purpose code in the payment info part. This entry can have the following values:

1 The code is **only** used as purpose code.
2 The code is **only** used as category purpose code.
3 The code is used as purpose code **and** as well as category purpose code.

65.4 Return codes

0 Everything was successful. The code was recognized. The results in the structure can be used.

-1 The code passed must have an exact length of 4 characters.

-2 The purpose code was not founded. No data was provided.

-999 The API wasn't initialized yet. Please first call the function SepaTools_Init.

66 SepaTools_SearchPurposeCodes

66.1 Purpose of the function

With this function, you can call all available purpose codes. The available purpose codes are the same as in the ISO External Code Sets listed.

To get all available codes, you have to call this function repeated until you get a function result not equal zero (Result <> 0).

66.2 Function call

int SepaTools_SearchPurposeCodes(*int First, PurposeStruct *Purpose, int Flag, int Typ)

66.3 Parameters

First Please provide here a value (pointer) greater than 0 (1 or greater) if you call this function the first time. The function will be initialized with this. After the first call, the function will set this value internal to the value of 0 (zero).

PurposeStruct In this structure the results of the function are provided..

struct PurposeStruct

{	int	Prio	Priority of the purpose code [1)]
	int	Typ	The type of the purpose code [2)]
	char	Code[4+1]	The 4 characters long purpose code
	char	Klasse[30+1]	The classification of the purpose code Codes
	char	Name[60+1]	The original term of the code
	char	Deutsch[60+1]	The German translation of the code (if available)
	char	Reserve[100]	Reserved for later use
}			

Flag Please provide a Flag which means how a missing German translation is to interpret. The following values are valid:

0 By a missing translation, the value **N/A** is provided.
1 By a missing translation, the string “Keine Übersetzung” is displayed.
2 The German translation is provided with the original English term.

Typ Please provide a value which indicates the kinds of codes which will be read. The following values are possible.

1 **Only** purpose codes will be read.
2 **Only** the category purpose codes are read.
3 Purpose codes **and** category purpose codes are read.

Additional notes:

1) With this code the “subjective” importance of the code is indicated. the following values are possible.

   0 This purpose code is used frequently, i.e. BENE or SALA etc.
   1 The other purpose codes contain the value 1.

2) Purpose codes are used in the transaction part as well the category purpose code in the payment info part. This entry can have the following values:

   1 The code is **only** used as purpose code.
   2 The code is **only** used as category purpose code.
   3 The code is used as purpose code **and** as well as category purpose code.

66.4 Return codes

0 Everything was successful. The code was recognized. The results in the structure can be used.

-1 No more codes are available.

-2 The database Sepa.dat couldn’t be open.

-3 The flag for the missing translation must have a value of 0, 1 or 2.

-4 The type of the code must have a value of 1, 2 or 3.

67 SepaTools_GetReasonCode

67.1 Purpose of the function

With this function, you can test a reason code if it exists. With an existing code, the structure returns additional information's of the code.

The available reason codes are the same as in the ISO External Code Sets listed.

67.2 Function call

int SepaTools_GetReasonCode(const char *Code, ReasonStruct *Reason)

67.3 Parameters

Code Please pass a pointer to the desired reason code (i.e. BE05, AM04 etc.). The code must be null terminated, 4 characters long.

Reasone Please pass a pointer to the structure ***ReasonStruct***. The structure is shown in then following.

ReasonStruct In this structure the results of the function are provided..

```
struct   ReasonStruct

{   char    Code[4+1]       The 4 characters long reason code
    char    TSErg[3+1]      An optional value for then text key extension 1)
    char    Name[100+1]     The original term of the code
    char    Reserve[100]    Reserved for later use
}
```

Additional notes:

1) For some less codes there is an equivalent corresponding value of the former (old) DTA format.

67.4 Return codes

0 Everything was successful. The code was recognized. The results in the structure can be used.

-1 The code passed must have an exact length of 4 characters.

-2 The reason code was not founded. No data was provided.

-999 The API wasn't initialized yet. Please first call the function SepaTools_Init.

68 SepaTools_SearchReasonCodes

68.1 Purpose of the function

With this function, you can call all available reason codes. The available reason codes are the same as in the ISO External Code Sets listed.

To get all available codes, you have to call this function repeated until you get a function result not equal zero (Result <> 0).

68.2 Function call

int SepaTools_SearchReasonCodes(*int First, ReasonStruct *Reason, int Typ)

68.3 Parameters

First Please provide here a value (pointer) greater than 0 (1 or greater) if you call this function the first time. The function will be initialized with this. After the first call, the function will set this value internal to the value of 0 (zero).

ReasonStruct In this structure the results of the function are provided..

```
struct    ReasonStruct

{   char    Code[4+1]       The 4 characters long reason code
    char    TSErg[3+1]      The optional text key extension 1)
    char    Name[100+1]     The original term of the code
    char    Reserve[100]    Reserved for later use
}
```

Typ In the External Code Sets there are different reason code defined. Over all there are more than 350 codes. Some reason codes can occur in different business transactions.

The file with the External Code Sets you can also find at https://sepa-tools.de and there under the menu item “Weitere Infos”.

The value of **Typ** is bit coded. With this coding you determine what codes will be listed.

The following table shows exemplary the constants which you can use to determine the desired values.

Constant	*Value*	*Original name within the External Code Sets with displaying the Excel sheet number*
Reason_None	0	All reason codes will be provided
Reason_Mandat	1	8-MandateReason
Reason_Return	2	13-ReturnReason
Reason_Reversal	4	14-ReversalReason
Reason_Status	8	16-StatusReason

Constant	*Value*	*Original name within the External Code Sets with displaying the Excel sheet number*
Reason_Verification	16	18-VerificationReason
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

Additional notes:

1) For some less codes there is an equivalent corresponding value of the former (old) DTA format.

68.4 Return codes

0 Everything was successful. The code was recognized. The results in the structure can be used.

-1 No more reason codes are available. No data was provided.

-2 The database Sepa.dat can't opened.

-999 The API wasn't initialized yet. Please first call the function SepaTools_Init.

69 SepaTools_AddUserBIC

69.1 Purpose of the function

With this function it is possible to deposit bank identifier codes in the User-Database which are divergent to the stock of BLZ of the deutsche Bundesbank or the Österreichische Nationalbank. When converting bank identifier codes in BICs, the divergent BIC is used.

This saved data only is used for converting of BLZ und bank account number in BIC und IBAN.

In case of those functions:

SepaTools_CreateXML (and the according functions)
SepaTools_ConvertBLZKonto
SepaTools_ReadDTA (and the according functions).

69.2 Function call

int SepaTools_AddUserBIC(const char *BLZ, const char *BIC)

69.3 Parameters

BLZ	Please deliver a pointer to the characters chain (string), which includes the bank routing number (8 characters for Germany and 5 for Austria)
BIC	Please deliver a pointer to the BIC, which has to be used, when the stated bank routing number should be converted.

69.4 Return codes

0	Everything was successful.
-1	The delivered bank routing number is formal wrong, length <>5 or <>8.
-2	The delivered BIC is formal wrong, length <>8 or length <>11.
-5	The entry couldn't be written in the database, because it is already included in the database.
-9	The processing of the User-Data wasn't initialized. Please proof the value for UserPath at the function function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

70 SepaTools_DelUserBIC

70.1 Purpose of the function

With this function, a combination out of bank routing number and BIC is deleted out of the data-base for the User-Data.

70.2 Function call

int SepaTools_DelUserBIC(const char *BLZ)

70.3 Parameters

BLZ	Please deliver a pointer to the characters string, which includes the bank routing number (8 characters for Germany and 5 for Austria)

70.4 Return codes

0	Everything was successful.
-1	The delivered bank identifier code is formal wrong, length <>5 or <>8.
-9	The processing of the User-Data wasn't initialized. Please proof the value for UserPath at the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

71 SepaTools_GetUserBIC

71.1 Purpose of the function

With this function the BIC which is saved for the corresponding bank routing number can be read out.

71.2 Function call

int SepaTools_GetUserBIC(const char *BLZ, char *BIC)

71.3 Parameter

BLZ	Please deliver a pointer to the characters string, which includes the bank routing number (8 characters for Germany and 5 for Austria).
BIC	In this variable the corresponding BIC is returned. Please note: The memory area has to be big enough and has to save at least 11+1 character (incl. the ending NULL)

71.4 Return codes

0	Everything was successful.
-1	The delivered bank routing number is formal wrong, length <>5 or <>8.
-6	No entry was found, for the stated bank routing number.
-7	It wasn't possible to read out the entry out of the database (because of technical reasons).
-9	The processing of the User-Data wasn't initialized. Please proof the value for UserPath at the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

72 SepaTools_FindUserBIC

72.1 Purpose of the function

This function serves for reading the database. A repeatedly call of the function until the return code is unequal 0 has as a result the reading out of all combinations of bank routing number and BIC.

72.2 Function call

int SepaTools_FindUserBIC(char *BLZ, char *BIC, int *First)

72.3 Parameters

BLZ	In this variable the read BLZ is passed (8 characters for Germany and 5 for Austria). The variable has to be big enough an has to save at least 8+1 characters.
BIC	In this variable the corresponding BIC is returned. Please note: The memory area has to be big enough and has to save at least 11+1 character (incl. the ending NULL)
First	Please state a value > 0, if you call the function for the first time. With this sequence the variables are initialized internally. After the first call of the function, the value is set to 0 by the API automatically.

72.4 Return codes

0	Everything was successful.
-6	You have reached the end of the database. there are no more entry's available.
-7	It wasn't possible to read out the entry out of the database (because of technical reasons).
-9	The processing of the User-Data wasn't initialized. Please proof the value for UserPath at the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

73 SepaTools_AddUserIBAN

73.1 Purpose of the function

With this function it is possible to deposit bank (8 characters for Germany and 5 for Austria) and corresponding bank account numbers in the User-Data for which a divergent IBAN should be established. Here can also be deposited a divergent BIC.

A divergent BIC only influences this combination of bank (8 characters for Germany and 5 for Austria)und bank account number. Do you want to deposit a divergent BIC for bank identifier code global, please use the function SepaTools_AddUserBIC.

The data saved on this way, only are used for conversion of BLZ and Bank account number in BIC and IBAN. This is true by the following functions:

SepaTools_CreateXML (and the corresponding functions)
SepaTools_ConvertBLZKonto
SepaTools_ReadDTA (and the corresponding functions).

73.2 Function call

int SepaTools_AddUserIBAN(const XMLConvertStruct *ConvertStruct)

73.3 Parameters

ConvertStruct In this structure the values are delivered. All fields have to be filled except the BIC. This field has only be filled if for the combination of delivered bank routing code and bank account number a divergent BIC should be used.

struct XMLConvertStruct

{	char	BLZ[8+1]	Bank routing code (8 characters for Germany and 5 for Austria), which has to be used.
	char	Konto[11+1]	Bank account number, which has to be used..
	char	BIC[11+1]	The BIC which has to be replaced (optional).
	char	IBAN[35+1]	The IBAN which has to be replaced
}			

73.4 Return codes

0	Everything was successful.
-1	The delivered bank identifier code is formal wrong, length <>5 or <>8.
-2	The delivered BIC is formal wrong, length <>8 or length <>11.
-3	The delivered bank account number is formal wrong. The numerical value of the bank account number is smaller than 10.
-4	Ether there was no IBAN, or an IBAN with a invalid test digit delivered.

-5	The entry couldn’t be written in the database, because it is already included in the database.
-9	The processing of the User-Data wasn’t initialized. Please proof the value for UserPath at the function SepaTools_Init.
-999	The API wasn’t initialized yet. Please first call the function SepaTools_Init.

74 SepaTools_DelUserIBAN

74.1 Purpose of the function

With this function the entry which is displayed by the combination of bank routing code and bank account number in the database is deleted.

74.2 Function call

int SepaTools_DelUserIBAN(const XMLConvertStruct *ConvertStruct)

74.3 Parameter

ConvertStruct In this structure the values are delivered. It is adequate if bank routing code and bank account number are delivered. BIC and IBAN aren't necessary for this function and aren't proofed.

struct XMLConvertStruct

{	char	BLZ[8+1]	Bank routing code (8 characters for Germany and 5 for Austria), which should be used.
	char	Konto[11+1]	Bank account number, which should be used.
	char	BIC[11+1]	No entry necessary.
	char	IBAN[35+1]	No entry necessary.
}			

74.4 Return codes

0	Everything was successful.
-1	The delivered bank identifier code is formal wrong, length <>5 or <>8.
-3	The delivered bank account number is formal wrong. The numerical value of the bank account number is smaller than 10.
-9	The processing of the User-Data wasn't initialized. Please proof the value for UserPath at the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

75 SepaTools_GetUserIBAN

75.1 Purpose of the function

With this function a data record with bank routing code, bank account number, replaceable IBAN and maybe replaceable BIC is read out of the database.

75.2 Function call

int SepaTools_GetUserIBAN(XMLConvertStruct *ConvertStruct)

75.3 Parameter

ConvertStruct In this structure the values are passed and the read values are returned. It is adequate if bank identifier code and bank account number are delivered. BIC and IBAN are returned,

struct XMLConvertStruct

{	char	BLZ[8+1]	Bank routing code (8 characters for Germany and 5 for Austria), which should be used.
	char	Konto[11+1]	Bank account number, which should be used.
	char	BIC[11+1]	The saved BIC.
	char	IBAN[35+1]	The saved IBAN.
}			

75.4 Return codes

0	Everything was successful.
-1	The delivered bank routing code is formal wrong, length <>5 or <>8.
-3	The delivered bank account number is formal wrong. The numerical value of the bank account number is smaller than 10.
-6	For the stated Bank routing code and the stated Bank account number no entry in the database was found.
-7	It wasn't possible to read out the entry out of the database (because of technical reasons).
-9	The processing of the User-Data wasn't initialized. Please proof the value for UserPath at the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

76 SepaTools_FindUserIBAN

76.1 Purpose of the function

With this function it is possible to read out the database. When you call the function repeatedly until the return code = 0, you can read out all combinations of bank identifier code, bank account number, IBAN and BIC.

76.2 Function call

int SepaTools_FindUserIBAN(const XMLConvertStruct *ConvertStruct , int *First)

76.3 Parameters

ConvertStruct In this structure, the read values are returned. It is not necessary to deliver values. The structure should be initialized with 0.

struct XMLConvertStruct

{	char	BLZ[8+1]	The saved bank routing code (8 characters for Germany and 5 for Austria), which should be used.
	char	Konto[11+1]	The saved bank account number.
	char	BIC[11+1]	The saved BIC.
	char	IBAN[35+1]	The saved IBAN.
}			

First The variable has to be deposit with a value of <> 0. With this the internal call sequence is initialized.

The API puts the variable to 0, when you call the function for the first time.

76.4 Return codes

0	Everything was successful.
-6	You reached the end of the database. There are no more entry's available in the database.
-7	It wasn't possible to read out the entry out of the database (because of technical reasons).
-9	The processing of the User-Data wasn't initialized. Please proof the value for UserPath at the function SepaTools_Init.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

77 SepaTools_GetTargetDatum

77.1 Purpose of the function

Book entries in the SEPA area can only be carried out on so called “Target days”. These are days on which the Europe wide Clearing system works. Target days are all days of the year except of Saturdays, Sundays, 25. December, 26. December, 1. January, 1. Mai, the Friday before eastern and the Monday of eastern.

The function is able to calculate the Friday before eastern and the Monday of eastern corresponding to the western churches (for orthodox churches and other calculations contact the manufacturer).

If you deliver a base date (e.g. the date of the day) to the function, the next target day will be calculated, with consideration of additional days (Parameter Skip=0). This is helpful in case of the execution date (direct debits) with consideration of the preliminary lead times.

Is the calculated date a non-target day, the date will be grossed up to the next target-day.

If you put the value of Skip to <> 0, in the calculation, the in the parameter “Tage” stated additional days the non-target days are jumped over. In case of calculating preliminary lead times the non-target-days are not counted.

The result is returned in a string.

For example:

Date of the day is: 19. December 2012. The date of the booking entry should be 6 days in future. The parameter Skip was deposit with 0. The next target day is: 27.12.2012.

Is the parameter Skip <> 0, while counting the 6 days the target days are not jumped over
The result is here: 31.12.2012

Take a calendar to test this function.

77.2 Function call

```
int SepaTools_GetTargetDatum(const char *Datum,
                             const int Tage,
                             const Int Skip,
                             char *TargetDatum)
```

77.3 Parameter

Datum		Please deliver a pointer to a characters string, which contains the origin date in the format DDMMYYYY.
Tage		Deliver the number of days, which should extend the origin date. Valid entries are between 0 und 1000. Do you use the value of 0, the function has as a result the same value independently of the value of Skip.
Skip	**0**	If there is a value of 0, the non-target days are not jumped over by calculating the date.

1 If there is a value of 1, the non-target days are jumped over at the counting of the additional days.

2 If there is a value of 2 the non-target days are jumped over at the counting of the additional days.

If the value of the parameter “date” is a non-target-day (e.g. Sunday), then the value of “date” will be increased to the next target-day.

TargetDatum This is the established date (format DDMMYYYY), which is a guaranteed target day. Please deliver a pointer to the memory area, which can save 9 characters (8 characters for the date with format DDMMYYYY and one character for the closing NULL).

77.4 Return codes

0	Everything was successful. The target date is established.
-1	The delivered origin date is not valid.
-2	The delivered origin date is smaller than the 1. January 2000. This is the start date for the function.
-3	The delivered value for days is < 0.
-4	The delivered value for days is > 1000.
-5	The target date could not be established. This would indicate a calculation error in the logic. Please contact the manufacturer.
-999	The API wasn’t initialized yet. Please first call the function SepaTools_Init.

78 SepaTools_IsTargetDatum

78.1 Purpose of the function

Book entries in the SEPA area can only be carried out on so called Target days. These are days on which the Europe wide Clearing system works. Target-days are all days of the year except of Saturdays, Sundays, 25. December, 26. December, 1. January, 1. Mai, Friday before eastern and the Monday of eastern.

The function is able to calculate the Friday before eastern and the Monday of eastern corresponding to the western churches (for orthodox churches and other calculations contact the manufacturer).

This function proofs if the delivered date is a valid target day.

78.2 Function call

int SepaTools_IsTargetDatum(const char *Datum)

78.3 Parameters

Datum	Please deliver a pointer to a characters string with the origin date in the format DDMMYYYY. It is proofed if it is a valid target day.

78.4 Return codes

0	Everything was successful. The target date is valid.
-1	The delivered origin date is not valid.
-2	The delivered origin date is smaller than the 1. January 2000. This is the start date for the function.
-3	The delivered date is no valid target date.
-999	The API wasn't initialized yet. Please first call the function SepaTools_Init.

79 Create pain.007 files

The API can create pain.007 XML files. Three functions are necessary for this. This file format is used for reversal direct debit payments (e.g. double presentation).

The support of pain.007 files through your payment provider is optional. In opposite to the processing of XML payment files, there is no obligation to support these files by the banks.

The logic is running as follows represents:

- function SepaTools_CreateXML007 — Starts and initializes the process.
- function SepaTools_WriteXML007 — Repeat this function call for every record with payment data.

  The limit for the number of calls is only the size of your memory area.
- function SepaTools_CloseXML007 — The process will be closed. The file is written to the specified data carrier.

Within this function will be held extensive plausibility checks.

80 SepaTools_CreateXML007

80.1 Purpose of the function

This function initializes then XML export. The corresponding parameters in the structure are passed.

80.2 Function call

int SepaTools_CreateXML007(XMLCreate007Struct *CreateStruct)

80.3 Parameter

CreateStruct Please pass a pointer to CreateStruct. The structure is shown below. All strings are terminated with a binary NULL. The length of the char array is therefore one byte longer as the actual text.

For all other strings it is checked internally if it is a valid character set for SEPA payments. Is it not, the invalid characters are removed. The corrected character string is returned in the structure.

Already this function checks whether in the specified path for the export file (the file which has to be emitted), can be written.

```
struct    XMLCreate007Struct

{   char    ExportPfad[255+1]       Drive and path for export (XML-file).
    char    XMLName[35+1]           The name of the XML-file to create 1).
    char    MsgId[35+1]             Message-Id for the XML-file 2).
    char    OrgMsgId[35+1]          The original Msg-Id of the reversal direct debit.
    char    EinreicherName[70+1]    Name of the presenter of the file (at least 3 chars).
    char    EinreicherBIC[11+1]     Optional the BIC of the submitter.
    char    Reserve[1000]           Reserved for later use.
}
```

Additional note:

1) Regardless of the passed file name extension or not passed file name extension the file name extension is always set to .xml.

   If an empty string (or the value Dummy.txt) is passed, the filename is managed internally by the API. This is necessary if the XML data should be exported to memory as a byte-array.

2) The Message-Id should be a clear identification of the possible XML file. If you pass an empty string here, it is automatically internally formed a Message-Id, and returned in the structure.

80.4 Return codes

0	Everything was successful
-1	The path for the export file is to short (< 2 chars).
-2	The specified directory for the export file does not exist.
-3	The data carrier for the export file is write-protected. The file can't be written.
-4	The name of the presenter is to short (less than 3 chars).
-5	For the process required temporary file could not be opened. Please check the parameter in the function "SepaTools_Init".
-6	The pathname for the export file is too long (>200 chars).
-7	The specified drive for the export file is not ready. (possibly no data carrier is inserted).
-8	The name for the XML-file is too short. (<3 chars).
-9	The function „SepaTools_CreateXML" was already called. Before a second call, the function „SepaTools_CloseXML" must be called. The functions to create XML-files are not „Multi-Thread" capable.

-10	No original Message-Id was passed. The Message-Id of the original direct debit file is mandatory.
-999	The API was not initializes. Please call first the function “SepaTools_Init“.

81 SepaTools_WriteXML007

81.1 Purpose of the function

On each call to this function exactly one record for a payment order is passed. There are extensive plausibility checks.

81.2 Function call

int SepaTools_WriteXML007(XMLWrite007Struct *WriteStruct)

81.3 Parameter

WriteStruct Pass here a pointer to the structure „WriteStruct“. The structure is shown below. All strings are passed null-terminated. The length of the char-array is always 1 byte longer as the actual text.

For all strings is checked internally whether it is a valid character set for SEPA payments. This is not the case, the invalid characters are removed. The corrected string is then returned in the structure

struct XMLWrite007Struct

{	char	PmInfoId[10+1]	Payment-Info-Id of the origin direct debit [1)]
	int	PmAnzahl	the count of the records within the Payment-Info-Block
	char	PmSumme[12+1]	The summery of the amount in this Payment-Info-Block [2)]
	int	Batch	Booking single entries of the reversal records [3)]
	char	Betrag[12+1]	The amount of the origin direct debit in Cent.
	char	Reason[4+1]	The reason of the reversal [4)]
	char	Aufdatum[8+1]	Execution date of the origin direct debit [5)]
	char	CI[35+1]	CI of the origin direct debit
	int	B2B	0=CORE direct debit, 1=B2B direct debit (origin)
	char	SequenceType[4+1]	Sequence of the origin direct debit [6)]
	char	MandatId[35+1]	Mandate Identification of the origin direct debit
	char	MandatDat[8+1]	Date of the mandate of the origin direct debit (TTMMJJJJ)
	char	Reserve[2000]	Reserved for later use
}			

Additional notes:

1) Please pass in the field the Payment-Info-Id of the origin direct debit. In the origin direct debit file can be more than one Payment-Info blocks. In this case it is the Payment-Info-Id of the corresponding recalled direct debit.

2) The summery of the amount within the origin Payment-Info-Block is shown in Cent! For e.g. then amount of 1.876,45 € is passed as a character string in the form 187645.

3) Only if you have an agreement with your file receiving bank, in the case of Batch=1 the reversals will be booked as single records on your statement of account.

   If you pass Batch=2 then then the booking will work as batch.

In the case of Batch=0 or Batch is empty the booking is according to the bank defaults.

4) Please pass here the reason of the reversal. The following values are allowed:

AM05 Duplicate submitting
MS02 Other (unknown) reason

5) The execution date of the origin direct debit in the form TTMMJJJJ.

81.4 Return codes

Note:

In the case of return code> = 0 (no error), in the return value, additional information are stored. This information is stored bitwise.

The bit assignment for the value >= 0 in the return value is as follows:

0	Everything was successful.
-1	The writing in the temporary file failed.
-2	The function „SepaTools_CreateXML“ was not yet called.
-101	No Payment-Information has been passed for the origin direct debit file.
-102	The count of the transactions in the origin direct debit file was not passed.
-103	The summery of the amounts in the origin direct debit file was not passed.
-104	No End-To-End-Id has been passed for the origin direct debit.
-105	No amount of the origin direct debit was passed.
-106	Nor reason for the reversal was passed.
-107	No execution date of the origin direct debit was passed.
-108	The execution date of the origin direct debit is in the future. This is not possible.
-109	No Credit identifier identification (CI) of the origin direct debit was passed.
-110	The value for B2B can only be 0 or 1.
-111	No sequence type for the origin direct debit was passed.
-112	The sequence type of the origin direct debit is not valid.
-113	No mandate identification was passed for the origin direct debit.
-114	For the mandate identification of the origin direct debit was no date passed or the date is invalid.

-115	The date of the mandate of the origin direct debit is in the future. This is not allowed.
-999	The API has not been initialized. First please call the function SepaTools_Init.

82 SepaTools_CloseXML007

82.1 Purpose of the function

With this function, the actual XML file is generated from the temporary file produced by SepaTools_WriteXML007 and written to the specified file.

82.2 Function Call

int SepaTools_CloseXML007(XMLCloseStruct *CloseStruct)

82.3 Parameters

CloseStruct Here a pointer is passed to the structure CloseStruct. The structure is shown below. The structure is passed empty and filled with information for the application by the API.

```
struct    XMLCloseStruct

{   int      Anzahl                Number of generated data records.
    char     SummeBetrag[12+1]     Total sum of amounts in the XML-file as a character
                                   string in cents.
}
```

82.4 Return codes

0	Everything was successful.
-1	The XML file could not be created/initialized. Please contact the manufacturer.
-2	When creating each record, there appeared an error. Please contact the manufacturer.
-3	When writing the XML file to disk an error has appeared.

The following return codes from -4 to -7 can only occur when operating in "memory mode" (no file name for the XML file has been passed). The XML file is not passed as a file. The content can be retrieved from the memory by the function SepaTools_XMLGetData (see page 68).

-4	There are no data available to output the XML file in the memory.
-5	The size of the XML data is greater than the maximum size allowed (500 MB).
-6	The memory area to transfer the data is invalid. Maybe there is not enough memory available.
-7	The output of the XML data in the memory area failed.
-8	No data records could be written, because all records had errors.
-9	The temporary file could not be opened.

-999	The API has not been initialized. Please first call to the function SepaTools_Init.

83 Create CGI files

The API can create CGI files. The CGI format is a much expanded XML format, similar as a SEPA format. With this way you can present foreign payments for payee outside of the SEPA area.

Please mind, that this format is actually supported by less banks. But the big banks support this format. Please ask your bank, if this format is supported.

Three functions are necessary for this. The logic is running as follows represents:

- function SepaTools_CreateCGI	Starts and initializes the process.
- function SepaTools_WriteCGI	Repeat this function call for every record with payment data. The limit for the number of calls is only the size of your memory area.
- function SepaTools_CloseCGI	The process will be closed. The file is written to the specified data carrier.

Within this function will be held extensive plausibility checks.

The features of this expanded XML format exceed the shown implementation far. This implementation was validated with NORDEA bank in Finland. To make required Extensions possible, we have a big reserved area.

84 SepaTools_CreateCGI

84.1 Purpose of the function

This function initializes then XML export. The corresponding parameters in the structure are passed.

84.2 Function call

int SepaTools_CreateCGI(CGICreateStruct *CreateStruct)

84.3 Parameter

CreateStruct Please pass a pointer to CreateStruct. The structure is shown below. All strings are terminated with a binary NULL. The length of the char array is therefore one byte longer as the actual text.

For all other strings it is checked internally if it is a valid character set. Is it not, the invalid characters are removed. The corrected character string is returned in the structure.

Already this function checks whether in the specified path for the export file (the file which has to be emitted), can be written.

struct CGICreateStruct

{	char	ExportPfad[255+1]	Drive and path for export (XML-file).
	int	Direct debit	Credit transfer(=0) or debit direct (=1). Is ignored
	char	XMLName[35+1]	The name of the XML-file to create [1)].
	char	MsgId[35+1]	Message-Id for the XML-file [2)].
	char	EinreicherName[70+1]	Name of the presenter of the file (at least 3 chars).
	char	EinreicherId[35+1]	Optional an Id for the presenter [3)]
	char	EinreicherShema[4+1]	Optional scheme name for the presenter, e.g. CUST [4)]
	int	ShortMsgId	Create a short EndToEnd-Id [5)]
	int	WhgSort	Sort data records by currency [6)]
	char	Reserve[992]	Reserved for later use.
}			

Additional note:

1) Regardless of the passed file name extension or not passed file name extension the file name extension is always set to .xml.

   If an empty string (or the value Dummy.txt) is passed, the filename is managed internally by the API. This is necessary if the XML data should be exported to memory as a byte-array.

2) The Message-Id should be a clear identification of the possible XML file. If you pass an empty string here, it is automatically internally formed a Message-Id, and returned in the structure.

3) Additional to the name of the presenter, the file receiving bank can request a special Id to identify the presenter.

You get this Id from your file receiving bank.

4) Additional to the optional Id for the presenter, you get from your bank a scheme name e.g. CUST.

5) Normally, according to the rulebook, an EndToEnd-Id can have up to 35 characters. Some banks request a short EndToEnd-Id with maximal 16 characters.

   If you pass the value “1” to this field, a short EndToEnd-Id with 16 only numeric characters will be created.

6) Normally the data records a not sorted by currency. The data records are in according to the currency mixed.

   If you pass the value 1 to this field, a sort by currency occurred. If the currency is changed, a new payment information block is created.

   Please ask your bank, if this is necessary.

84.4 Return codes

0	Everything was successful
-1	The path for the export file is to short (< 2 chars).
-2	The specified directory for the export file does not exist.
-3	The data carrier for the export file is write-protected. The file can’t be written.
-4	The name of the presenter is to short (less than 3 chars).
-5	For the process required temporary file could not be opened. Please check the parameter in the function “SepaTools_Init”.
-6	The pathname for the export file is too long (>200 chars).
-7	The specified drive for the export file is not ready. (possibly no data carrier is inserted).
-8	The name for the XML-file is too short. (<3 chars).
-9	The function „SepaTools_CreateXML“ was already called. Before a second call, the function „SepaTools_CloseXML” must be called. The functions to create XML-files are not „Multi-Thread“ capable!
-999	The API was not initializes. Please call first the function “SepaTools_Init“.

85 SepaTools_WriteCGI

85.1 Purpose of the function

On each call to this function exactly one record for a payment order is passed. There are extensive plausibility checks.

It can be mixed payment orders are passed by different customers. The orders are automatically sorted by customers, so that multiple payment transactions from the same customer, the orders internally automatically are sorted and collected.

85.2 Function call

int SepaTools_WriteGI(CGIWriteStruct *WriteStruct)

85.3 Parameter

WriteStruct Pass here a pointer to the structure „WriteStruct“. The structure is shown below. All strings are passed null-terminated. The length of the char-array is always 1 byte longer as the actual text.

For all strings is checked internally whether it is a valid character set. This is not the case, the invalid characters are removed. The corrected string is then returned in the structure

Banking expertise

The initiator of a payment is for credit transfer the payer of the order. For direct debit, the initiator is the recipient of the payment.

For credit transfer the recipient of the payment is the part get the payment. For direct debit, the recipient of the payment is who has to pay.

struct CGIWriteStruct

{	int	KontrollSumen	Control sums (Amount and countl) anlegen [1)]
	char	PmInfoId[10+1]	PaymentInfoId – Designation of the order [2)].
	char	AusfDatum[8+1]	Execution date in the form TTMMJJJJ
	char	AuftragName[70+1]	Name of the client of the payments
	char	AuftragNameAbw[70+1]	Ultimate name of the client of the payments
	char	AuftragLand[2+1]	ISO-country Id of the country of the client e.g. US
	char	AuftragAdrLine1_Str[35+1]	Address line 1 of the clients address [3)]
	char	AuftragAdrLine2_Ort[35+1]	Address line 2 of the clients address [3)]
	char	AuftragAdrHausNr[10+1]	Optional a building number of the address [3)]
	char	AuftragAdrPLZ[15+1]	Optional a ZIP-code of the address, z.B. NY 10023 [3)]
	int	AuftragAdrStruct	Unstruct. (Value=0) or structured (Value=1) address [3)]
	char	AuftragId[35+1]	Optional a client-Id for the payments [4)]
	char	AuftragSchema[4+1]	Optional a scheme name for the client-Id [5)]
	char	AuftragIBAN[35+1]	IBAN of the client account
	char	AuftragBIC[11+1]	BIC of the client account
	char	AuftragWhg[3+1]	ISO-currency code of the clients account, e.g. USD
	char	EndToEndId[10+1]	Id of the single payment [6)]

char	InstructionId[35+1]	Optional an Instruction-Id
int	BatchBooking	Batch-Booking Option [7)]
char	CatPurpose[4+1]	Optional a Category Purpose Code
char	Betrag[12+1]	Amount of the payment as string in cent.
char	Whg[3+1]	Currency of the payment
char	EmpfName[70+1]	Name of the payee
char	EmpfNameAbw[70+1]	Ultimate name of the payee
char	EmpfLand[2+1]	ISO-country code of the payee
char	EmpfAdrLine1_Str[35+1]	Address line 1of the address of the payee [3)]
char	EmpfAdrLine2_Ort[35+1]	Address line 2 of the address of the payee[3)]
char	EmpfAdrHausNr[10+1]	Optional a building number of the address [3)]
char	EmpfAdrPLZ[15+1]	Optional the ZIP-code of the address, e.g. NY 10023 [3)]
int	EmpfAdrStruct	Unstruct. (Value=0) or structured (Value=1) address [3)]
char	EmpfIBAN[35+1]	IBAN or account number of the payee [8)]
char	EmpfKontoCode[4+1]	Code for the account of the payee [12)]
char	EmpfBIC[11+1]	BIC of the payee [9)]
char	BankName[70+1]	Optional name of the receiving bank
char	BankLand[2+1]	ISO-country code of the bank of the payee
char	BankAdrLine1_Str[35+1]	Addressline 1of the address of the payee [3)]
char	BankAdrLine2_Ort[35+1]	Addressline 2 of the address of the payee [3)]
char	BankAdrHausNr[10+1]	Optional the building number of the address [3)]
char	BankAdrPLZ[15+1]	Optional the ZIP-code of the address, e.g. NY 10023 [3)]
int	BankAdrStruct	Unstruct. (Value=0) or structured (Value=1) Address [3)]
char	BankClearingSysId[5+1]	Optional a Bank Clearing Sys-Id [10)]
char	BankMemberId[35]+1	Bank Member-Id for addressing the bank [9)]
char	Purpose[4+1]	Optional a purpose-code
char	Zweck1[70+1]	First part of the purpose of the payment [11)]
char	Zweck2[70+1]	Second part of the purpose of the payment [11)]
char	StructZweck[35+1]	Reference number, e.g. RF81123453 [11)]
char	StructTyp[4+1]	Typ of the reference, e.g. SCOR [11)]
char	StructIssr[35+1]	Optional an Issuer, e.g. ISO [11)]
int	DoStruct	Use structured purpose (Value=1) [11)]
char	Reserve[3000]	Reserved for later use

}

Additional notes:

1) Inside of the payment information block control sums for the amount and the count of payments can be written. If this is necessary and valid, or is it not depends from the file receiving bank.

   Set this field to the value=1 if the control sums should be written.

2) To uniquely identify the initiator an internal ID is required. Please pass an up to 10-digit abbreviation for this ID. Internally, the abbreviation is extended with the date, time and a serial number.

   If you pass an empty string here, the complete identification will be automatically formed.

   As a result of a possible internal sorting the PmInfoId is formed at a later point in time in the context of calling “SepaTools_CloseCGI”, it could not be returned at this point in the structure. The actual results can be seen only in the generated XML file.

3) Address information can be passed in two different ways. If address information should be written in the XML file, a passed field “country” is mandatory.

   The address can be passed unstructured in two address lines or structured with the values street, building number, ZIP-code und city.

   The field “*AdrLin1_Str” is filled with the address line 1 or the street of the address. The field “*AdrLine2_Ort” is filled with the address line 2 or the city of the address.

   If the unstructured or the structured address is used, depends from the field “AdrStruct”. With the value 0 the unstructured and with the value the structured address is used.

4) Additional to the name of the client, the file receiving bank can request an optional client-Id to identify the client.

   This Id you get from your bank.

5) Additional to the optional client-Id the file receiving bank can request for an optional scheme-code, e.g. BANK.

6) To uniquely identify the payment order an internal ID is required. Please enter an up to 10-digit abbreviation for this ID. Internally, the abbreviation is then extended with the date, time and a serial number.

   If you pass an empty string here, the complete identification is created automatically.

   As a result of the possible internal sorting EndToEndId is formed only at a later time in the context of the call to “SepaTools_CloseCGI”, they it could not returned at this point to the in the structure. The actual result can be seen in the XML file.

7) The value in the field “BatchBooking” decides, if the single payments are booked as a batch or as single bookings on the account of the client.

   We have the following shaping's:

   0 The booking depends of the default settings of the bank
   1 Single payment booking
   2 Batch booking

   The effectiveness of that value depends from the settings in the bank. It can be, that this value did not affect.

8) In this field you can pass the IBAN or a local account number of the payee. Internally is checked if it is an IBAN or an account number. The correct handling of the XML-file is automatically.

9) The addressing of the bank of the payee can be with a BIC or a Clearing Member-Id. Only one of these two values may be handed over.

   In the case, that the BIC is invalid or the Member-Id is passed, the Clearing Member-Id has priority.

10) In connection with the Member-Id the bank can optional request a Clearing Sys-Id (e.g. BANK).

11) The purpose can be passed structured (like a reference number, e.g. RF566565654) or unstructured. The two lines of the unstructured purpose will added to one line in the XML file.

    The field “DosStruct” defines if the purpose is structured (Value=1) or unstructured (Value=0).

12) If you don’t use an IBAN to address the account of the payee, because national account number, it can be requested to pass an additional scheme for this account. This can be e.g. BBAN.

85.4 Return codes

0	Everything was successful.
-1	The writing in the temporary file failed.
-2	The function „SepaTools_CreateXML“ was not yet called.
-101	The name of the recipient was not passed, because have less than 3 characters.
-104	No amount was passed or the amount is less 0 (negative).
-153	There was ether no IBAN delivered or the IBAN is incorrect. The IBAN can’t be used.
-163	The execution date is invalid, or it is past.
-164	The BIC code of the initiator is not listed in the list of reachable banks. The BIC can't be used.
-401	No message-Id was passed.
-402	No BIC was passed for the client.
-403	The currency for the account of the client is invalid.
-404	No EndToEnd-Id was passed.
-405	The currency for the payment is unvalid.
-999	The API has not been initialized. First please call the function SepaTools_Init.

86 SepaTools_CloseCGI

86.1 Purpose of the function

With this function, the actual XML file is generated from the temporary file produced by SepaTools_WriteCGI and written to the specified file.

86.2 Function Call

int SepaTools_CloseCGI(XMLCloseStruct *CloseStruct)

86.3 Parameters

CloseStruct Here a pointer is passed to the structure CloseStruct. The structure is shown below. The structure is passed empty and filled with information for the application by the API.

struct XMLCloseStruct

{	int	Anzahl	Number of generated data records.
	char	SummeBetrag[12+1]	Total sum of amounts in the XML-file as a character string in cents.

}

86.4 Return codes

0	Everything was successful.
-1	The XML file could not be created/initialized. Please contact the manufacturer.
-2	When creating each record, there appeared an error. Please contact the manufacturer.
-3	When writing the XML file to disk an error has appeared.

The following return codes from -4 to -7 can only occur when operating in "memory mode" (no file name for the XML file has been passed). The XML file is not passed as a file. The content can be retrieved from the memory by the function SepaTools_XMLGetData.

-4	There are no data available to output the XML file in the memory.
-5	The size of the XML data is greater than the maximum size allowed (500 MB).
-6	The memory area to transfer the data is invalid. Maybe there is not enough memory available.
-7	The output of the XML data in the memory area failed.
-8	No data records could be written, because all records had errors.
-9	The temporary file could not be opened.
-999	The API has not been initialized. Please first call to the function SepaTools_Init.