L

UCBioBSP SDK

Union Community's Software Development Kit for Biometric Application

Programmer's Guide

Version 3.xx Rev 01

UNION COMMUNITY Co., Ltd.

Copyright © 2008, UNION COMMUNITY Co., Ltd.
All rights reserved.

USER License Agreement for Software Developer's Kit Designed by Union Community Co., Ltd

This agreement is a legal usage license agreement between Union Community Co., Ltd. and the user. If you do not agree with the terms and condition of the agreement, please return the product promptly. If you return the product, you will receive a refund.

1. Usage License
UNION COMMUNITY Co., Ltd. Grants licensee to use this SDK a personal, Limited, non-transferable, non-exclusive right to install and use one copy of the SDK on a single computer exclusively.
The software is considered 'being used' if it is stored in a computer's main or other storage device. The number of software copies will be determined by taking the greater number of the number of computers 'used' by the software and the number of computers with the software stored.
Licensee may use the SDK solely for developing, designing, and testing UNION software applications for use with UNION products ("Applications").

2. Right to Upgrade
If you have purchased the software by upgrading an older version, the usage license of the old version is transferred to the new version. However, you may only use the old version under the condition that the old and new versions are not running simultaneously. Therefore, you are prohibited from transferring, renting or selling the old version. You maintain the usage license for the program and ancillary files that are in the old version but not in the new version.

3. Assignment of License
If you wish to transfer the usage license of this software to a third party, you must first obtain a written statement indicating that the recipient agrees with this agreement. You must then transfer the original disk and all other program components, and all copies of the program must be destroyed. After the transfer is complete, you must notify UNION COMMUNITY Co., Ltd. to update the customer registration.
Licensee shall not rent, lease, sell or lend the software application developed using the SDK to a third party without UNION's prior written consent.
Licensee shall not copy and redistribute the SDK without UNION's prior written consent.
No other uses and/or distribution of the SDK or Sample Code are permitted without UNION's prior written consent. UNION reserves all rights not expressly granted to Licensee.

4. Copyright
All copyrights and intellectual properties of the software and its components belong to UNION COMMUNITY Co., Ltd. and these rights are protected under Korean and international copyright laws. Therefore, you may not make copies of the software other than for your backup purposes. In addition,

you may not modify the software other than for reverse-engineering purposes to secure compatibility. Finally, you may not modify, transform or copy any part of the documentation without written permission from UNION COMMUNITY. (If you're using a network product, you may copy the documentation in the amount of the number of users)

5. Installation

An individual user can install this software in his/her PCs at home and office, as well as in a mobile PC. However, the software must not be running from two computers simultaneously. A single product can be installed in two or more computers in one location, but one of those computers must have a usage rate of at least 70%. If another computer has a usage rate of 31% or higher, another copy of the software must be purchased.

6. Limitation of Warranty

UNION COMMUNITY Co., Ltd. guarantees that the CD-ROM and all components are free of physical damage for a year after purchase.
UNION DISCLAIMS ALL WARRANTIES NOT EXPRESSLY PROVIDED IN THIS AGREEMENT INCLUDING, WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE. If you find any manufacture defect within the warranty period, we will replace the product. You must be able to prove that the product has been purchased within a year to receive a replacement, but we will not replace a product damaged due to your mishandling or negligence. UNION COMMUNITY Co., Ltd. does not guarantee that the software and its features will satisfy your specific needs, and is not liable for any consequential damages arising out of the use of this product.

7. Liabilities

UNION COMMUNITY Co., Ltd. is not liable for any verbal, written or other agreements made by third parties, including product suppliers and dealers.

8. Termination

This agreement is valid until the date of termination. However, the agreement shall terminate automatically if you damage the program or its components, or fail to comply with the terms described in this agreement.

9. Customer Service

UNION COMMUNITY Co., Ltd. makes every effort to provide registered customers with technical assistance and solutions to problems regarding software applications under certain system environments. When a customer submits a suggestion about any inconvenience or anomaly experienced during product usage, UNION COMMUNITY Co., Ltd. will take corrective action and notify the customer of the result.

10. General Terms

You acknowledge that you have read, understood and agree with the terms of this agreement. You also recognize the fact that this agreement has precedence over user agreements of older versions, past order agreements, advertisement notifications and/or other written agreements.

11. Contact

If you have any questions about this agreement, please contact UNION COMMUNITY Co., Ltd. via telephone, fax or e-mail.

Table of Contents

1. Overview 20

1.1. Special Features of SDK 21
1.2. Provided Module 22
1.3. Development Model 23
1.4. Fingerprint Data Structure 24
1.4.1. Format 24
1.4.2. Header 24
1.4.3. Data Block 26
1.5. Terminology Description 27

2. Installation 28

2.1. System Requirements 28
2.2. Installing 29
2.3. Description of Files and Folders to Be Installed 36
2.3.1. Windows System32 folder 36
2.3.2. GAC (Global Assembly Cache) folder 36
2.3.3. (Installation folder)\Inc 36
2.3.4. (Installation folder)\Lib 37
2.3.5. (Installation folder)\Bin 37
2.3.6. (Installation folder)\dotNET 37
2.3.7. (Installation folder)\dotNet\Setup 38
2.3.8. (Installation folder)\Samples 38
2.3.9. (Installation folder)\Skins 38

3. Programming by using DLL 39

3.1. Function Call Structure 39
3.2. Initialization and Termination 40
3.2.1. Initializing 40
3.2.2. Terminating 40

3.3. Basic Setting 41
3.3.1. Obtaining SDK version 41
3.3.2. Obtaining basic setting values & Setting up new values 41
3.4. Using Device 44
3.4.1. Obtaining device list 44
3.4.2. Opening device 45
3.4.3. Closing device 45
3.4.4. Obtaining device information 46
3.5. Understanding FIR Data 47
3.5.1. The type of FIR 47
3.5.2. The use of FIR 47
3.5.3. Releasing FIR memory 49
3.5.4. Conversion of FIR 49
3.6. Registering Fingerprint 51
3.6.1. New fingerprint registration & Existing fingerprint modification 51
3.6.2. Payload designation 51
3.6.3. Auto Detect 51
3.6.4. Example of use 51
3.7. Acquiring Fingerprint 53
3.7.1. Fingerprint acquisition 53
3.7.2. Example of use 53
3.8. Authenticating Fingerprint 54
3.8.1. Authenticating live fingerprint with registered fingerprint 54
3.8.2. Authenticating already acquired fingerprint with registered fingerprint 54
3.8.3. Obtaining Payload data 56
3.9. Using FastSearch (1:N Authentication) 57
3.9.1. Initialization and Termination 57
3.9.2. Obtaining basic setting values & Setting new values 57
3.9.3. Creating DB 57
3.9.4. Memory DB management 58
3.9.5. Authenticating 1:N 59
3.10. Converting FIR Data 61
3.10.1. Extracting template data from FIR data 61
3.10.2. Creating FIR Handle using template data 61
3.10.3. Conversion between template data 63

3.10.4. Extracting raw image from Audit FIR data 63
3.10.5. Creating Audit FIR Handle using raw image 64
3.11. Setting UI 65
3.11.1. Loading skin file 65
3.11.2. Changing UI property 65
3.11.3. Using Callback 67
3.11.4. Example of use 69
3.12. Using the Smart Card 70
3.12.1. Outline of the smart card 70
3.12.2. Switching on/off RF power of smart card 73
3.12.3. Reading serial number of smart card 74
3.12.4. Reading & Writing block value 74
4. Programming by using COM 76
4.1. Outline of COM Use 76
4.1.1. Registration of COM 76
4.2. Initialization and Termination 77
4.2.1. Initializing 77
4.2.2. Terminating 77
4.2.3. Lower interface declaration 77
4.3. Basic Setting 80
4.3.1. Obtaining SDK version 80
4.3.2. Obtaining basic setting values & Setting new values 80
4.4. Using Device 82
4.4.1. Obtaining device list 82
4.4.2. Opening device 83
4.4.3. Closing device 84
4.4.4. Obtaining device information 84
4.5. Understanding FIR Data 86
4.5.1. The type of FIR 86
4.5.2. The use of FIR 86
4.5.3. Releasing FIR memory 86
4.6. Registering Fingerprint 87
4.6.1. New fingerprint registration & Existing fingerprint modification 87

4.6.2. Payload designation 87
4.6.3. Auto Detect 87
4.6.4. Example of use 87
4.7. Acquiring Fingerprint 89
4.7.1. Fingerprint acquisition 89
4.7.2. Example of use 89
4.8. Authenticating Fingerprint 91
4.8.1. Authentication live fingerprint wit registered fingerprint 91
4.8.2. Authenticating already acquired fingerprint with registered fingerprint 91
4.8.3. Obtaining Payload data 93
4.9. Using FastSearch (1:N Authentication) 94
4.9.1. Initialization and Termination 94
4.9.2. Obtaining basic setting values & Setting up new values 94
4.9.3. Creating DB 95
4.9.4. Memory DB management 96
4.9.5. Authenticating 1:N 96
4.10. Converting FIR Data 99
4.10.1. Extracting template data from FIR data 99
4.10.2. Creating FIR using template data 99
4.10.3. Extracting raw image from Audit FIR data 100
4.11. Setting UI 102
4.11.1. Loading skin file 102
4.11.2. Changing UI property 102
4.11.3. Using Callback Event Handler 103
4.12. Using Smart Card 105
4.12.1. Initialization and Termination 105
4.12.2. Switching on/off RF power of smart card 105
4.12.3. Reading serial number of smart card 106
4.12.4. Reading & Writing block value 107
5. API Reference for DLL 109
5.1. Type definitions 109
5.1.1. Basic types 109
■ UCBioAPI_SINT8 / UCBioAPI_SINT16 / UCBioAPI_SINT32 109
■ UCBioAPI_UINT8 / UCBioAPI_UINT16 / UCBioAPI_UINT32 / UCBioAPI_UINT64 109

- UCBioAPI_SINT / UCBioAPI_UINT 109
- UCBioAPI_VOID_PTR 109
- UCBioAPI_BOOL 109
- UCBioAPI_CHAR / UCBioAPI_CHAR_PTR 109
- UCBioAPI_NULL 109
- UCBioAPI_HWND 109

5.1.2. General types 110

- UCBioAPI_FIR_VERSION 110
- UCBioAPI_VERSION 110
- UCBioAPI_FIR_DATA_TYPE 110
- UCBioAPI_FIR_PURPOSE 110
- UCBioAPI_FIR_QUALITY 111
- UCBioAPI_FIR_PRIVILEGE 111
- UCBioAPI_FIR_DATE 112
- UCBioAPI_FIR_UUID_INFO 112
- UCBioAPI_OPTIONAL_DATA_TYPE 112
- UCBioAPI_FIR_OPTIONAL_DATA 113
- UCBioAPI_FIR_HEADER 114
- UCBioAPI_FIR_DATA 115
- UCBioAPI_FIR_FORMAT 115
- UCBioAPI_FIR 115
- UCBioAPI_FIR_PAYLOAD 115
- UCBioAPI_HANDLE / UCBioAPI_HANDLE_PTR 116
- UCBioAPI_FIR_HANDLE / UCBioAPI_FIR_HANDLE_PTR 116
- UCBioAPI_FIR_SECURITY_LEVEL 116
- UCBioAPI_TEMPLATE_FORMAT 117
- UCBioAPI_INIT_INFO_0 117
- UCBioAPI_DEVICE_ID 118
- UCBioAPI_DEVICE_NAME 119
- UCBioAPI_DEVICE_INFO_0 119
- UCBioAPI_DEVICE_INFO_EX 120
- UCBioAPI_RETURN 121
- UCBioAPI_FIR_TEXTENCODE 121
- UCBioAPI_INPUT_FIR_FORM 122
- UCBioAPI_INPUT_FIR 122
- UCBioAPI_WINDOW_CALLBACK_PARAM_0 123
- UCBioAPI_WINDOW_CALLBACK_PARAM_1 123
- UCBioAPI_WINDOW_CALLBACK_0 / UCBioAPI_WINDOW_CALLBACK_1 124
- UCBioAPI_CALLBACK_INFO_0 / UCBioAPI_CALLBACK_INFO_1 124
- UCBioAPI_WINDOW_STYLE 125
- UCBioAPI_WINDOW_OPTION 126

- UCBioAPI_WINDOW_OPTION_2 127
- UCBioAPI_TEMPLATE_TYPE 128
- UCBioAPI_FINGER_ID 129
- UCBioAPI_MATCH_OPTION_0 129

5.1.3. Export/Import functions related types 131

- UCBioAPI_TEMPLATE_BLOCK 131
- UCBioAPI_FINGER_BLOCK 131
- UCBioAPI_EXPORT_DATA 132
- UCBioAPI_IMAGE_DATA 132
- UCBioAPI_AUDIT_DATA 133
- UCBioAPI_EXPORT_AUDIT_DATA 134

5.1.4. FastSearch functions related types 135

- UCBioAPI_FASTSEARCH_INIT_INFO_0 135
- UCBioAPI_FASTSEARCH_INIT_INFO_0 136
- UCBioAPI_FASTSEARCH_SAMPLE_INFO 136
- UCBioAPI_FASTSEARCH_CALLBACK_PARAM_0 137
- UCBioAPI_FASTSEARCH_CALLBACK_0 137
- UCBioAPI_FASTSEARCH_CALLBACK_INFO_0 138

5.1.5. SmartCard functions related types 139

- UCBioAPI_SC_USE_KEY_A / UCBioAPI_SC_USE_KEY_B 139
- UCBioAPI_SC_LED_TOGGLE / UCBioAPI_SC_LED_NOT_TOGGLE 139

5.2. Error Definitions 140

5.2.1. Success 140

- UCBioAPIERROR_NONE 140

5.2.2. General error definitions 141

- UCBioAPIERROR_INVALID_HANDLE 141
- UCBioAPIERROR_INVALID_POINTER 141
- UCBioAPIERROR_INVALID_TYPE 141
- UCBioAPIERROR_FUNCTION_FAIL 141
- UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED 141
- UCBioAPIERROR_ALREADY_PROCESSED 142
- UCBioAPIERROR_EXTRACTION_OPEN_FAIL 142
- UCBioAPIERROR_VERIFICATION_OPEN_FAIL 142
- UCBioAPIERROR_DATA_PROCESS_FAIL 142
- UCBioAPIERROR_MUST_BE_PROCESSED_DATA 142
- UCBioAPIERROR_INTERNAL_CHECKSUM_FAIL 143
- UCBioAPIERROR_ENCRYPTED_DATA_ERROR 143
- UCBioAPIERROR_UNKNOWN_FORMAT 143
- UCBioAPIERROR_UNKNOWN_VERSION 143
- UCBioAPIERROR_VALIDITY_FAIL 144
- UCBioAPIERROR_INVALID_TEMPLATESIZE 144

- UCBioAPIERROR_INVALID_TEMPLATE 144
- UCBioAPIERROR_EXPIRED_VERSION 144
- UCBioAPIERROR_INVALID_SAMPLESPERFINGER 144
- UCBioAPIERROR_UNKNOWN_INPUTFORMAT 145
- UCBioAPIERROR_INVALID_PARAMETER 145
- UCBioAPIERROR_FUNCTION_NOT_SUPPORTED 145

5.2.3. Initialization related error definitions 146

- UCBioAPIERROR_INIT_MAXFINGERSFORENROLL 146
- UCBioAPIERROR_INIT_NECESSARYENROLLNUM 146
- UCBioAPIERROR_INIT_SAMPLESPERFINGER 146
- UCBioAPIERROR_INIT_SECULEVELFORENROLL 146
- UCBioAPIERROR_INIT_SECULEVELFORVERIFY 146
- UCBioAPIERROR_INIT_SECULEVELFORIDENTIFY 146

5.2.4. Device related error definitions 147

- UCBioAPIERROR_DEVICE_OPEN_FAIL 147
- UCBioAPIERROR_INVALID_DEVICE_ID 147
- UCBioAPIERROR_WRONG_DEVICE_ID 147
- UCBioAPIERROR_DEVICE_ALREADY_OPENED 147
- UCBioAPIERROR_DEVICE_NOT_OPENED 147
- UCBioAPIERROR_DEVICE_BRIGHTNESS 148
- UCBioAPIERROR_DEVICE_CONTRAST 148
- UCBioAPIERROR_DEVICE_GAIN 148

5.2.5. User interface related error definitions 149

- UCBioAPIERROR_USER_CANCEL 149
- UCBioAPIERROR_USER_BACK 149
- UCBioAPIERROR_CAPTURE_TIMEOUT 149
- UCBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS 149
- UCBioAPIERROR_ENROLL_EVENT_PLACE 149
- UCBioAPIERROR_ENROLL_EVENT_HOLD 149
- UCBioAPIERROR_ENROLL_EVENT_REMOVE 149
- UCBioAPIERROR_ENROLL_EVENT_PLACE_AGAIN 149
- UCBioAPIERROR_ENROLL_EVENT_PROCESS 150
- UCBioAPIERROR_ENROLL_EVENT_MATCH_FAILED 150

5.2.6. FastSearch related error definitions 151

- UCBioAPIERROR_FASTSEARCH_INIT_FAIL 151
- UCBioAPIERROR_FASTSEARCH_SAVE_DB 151
- UCBioAPIERROR_FASTSEARCH_LOAD_DB 151
- UCBioAPIERROR_FASTSEARCH_UNKNOWN_VER 151
- UCBioAPIERROR_FASTSEARCH_IDENTIFY_FAIL 151
- UCBioAPIERROR_FASTSEARCH_DUPLICATED_ID 152
- UCBioAPIERROR_FASTSEARCH_IDENTIFY_STOP 152

- UCBioAPIERROR_FASTSEARCH_NOUSER_EXIST 152

5.2.7. Optional value related error definitions 153
- UCBioAPIERROR_OPTIONAL_UUID_FAIL 153
- UCBioAPIERROR_OPTIONAL_PIN1_FAIL 153
- UCBioAPIERROR_OPTIONAL_PIN2_FAIL 153
- UCBioAPIERROR_OPTIONAL_SITEID_FAIL 153
- UCBioAPIERROR_OPTIONAL_EXPIRE_DATE_FAIL 153

5.2.8. SmartCard related error definitions 154
- UCBioAPIERROR_SC_FUNCTION_FAILED 154
- UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE 154
- UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE 154

5.3. API References 155

5.3.1. Basic API 155
- UCBioAPI_Init 155
- UCBioAPI_Terminate 156
- UCBioAPI_GetVersion 157
- UCBioAPI_GetInitInfo 158
- UCBioAPI_SetInitInfo 159
- UCBioAPI_SetSkinResource 160
- UCBioAPI_SetCaptureQuality 160

5.3.2. Device related API 161
- UCBioAPI_EnumerateDevice 161
- UCBioAPI_OpenDevice 162
- UCBioAPI_CloseDevice 163
- UCBioAPI_GetDeviceInfo 164
- UCBioAPI_SetDeviceInfo 165
- UCBioAPI_AdjustDevice 166
- UCBioAPI_GetOpenedDeviceID 167
- UCBioAPI_CheckFinger 168
- UCBioAPI_SetAutoDetect 168

5.3.3. Memory related API 170
- UCBioAPI_GetFIRFromHandle 170
- UCBioAPI_GetExtendedFIRFromHandle 171
- UCBioAPI_GetHeaderFromHandle 172
- UCBioAPI_GetExtendedFIRFromHandle 173
- UCBioAPI_GetTextFIRFromHandle 174
- UCBioAPI_GetExtendedTextFIRFromHandle 175
- UCBioAPI_FreeFIRHandle 176
- UCBioAPI_FreeFIR 177
- UCBioAPI_FreeTextFIR 178
- UCBioAPI_FreePayload 179

5.3.4. Core API 180
- UCBioAPI_Capture 180
- UCBioAPI_Process 182
- UCBioAPI_CreateTemplate 183
- UCBioAPI_VerifyMatch 185
- UCBioAPI_VerifyMatchEx 186
- UCBioAPI_Enroll 188
- UCBioAPI_Verify 190

5.3.5. Data conversion API 192
- UCBioAPI_FIRToTemplate 192
- UCBioAPI_TemplateToFIR 194
- UCBioAPI_TemplateToFIREx 196
- UCBioAPI_ConvertTemplateData 198
- UCBioAPI_ImportDataToFIR 200
- UCBioAPI_ImportDataToFIREx 202
- UCBioAPI_AuditFIRToImage 204
- UCBioAPI_ImageToAuditFIR 205
- UCBioAPI_FreeData 206
- UCBioAPI_FreeExportData 207
- UCBioAPI_FreeExportAuditData 208

5.3.6. FastSearch API 209
- UCBioAPI_InitFastSearchEngine 209
- UCBioAPI_TerminateFastSearchEngine 210
- UCBioAPI_GetFastSearchInitInfo 211
- UCBioAPI_SetFastSearchInitInfo 212
- UCBioAPI_AddFIRToFastSearchDB 213
- UCBioAPI_RemoveFpFromFastSearchDB 215
- UCBioAPI_RemoveUserFromFastSearchDB 216
- UCBioAPI_IdentifyFIRFromFastSearchDB 217
- UCBioAPI_ClearFastSearchDB 219
- UCBioAPI_SaveFastSearchDBToFile 220
- UCBioAPI_LoadFastSearchDBToFile 221
- UCBioAPI_GetFpCountFromFastSearchDB 222
- UCBioAPI_GetFpInfoFromFastSearchDB 223
- UCBioAPI_CheckFpExistInFastSearchDB 224

5.3.7. SmartCard API 225
- UCBioAPI_SC_RFPowerOn 225
- UCBioAPI_SC_RFPowerOff 226
- UCBioAPI_SC_RFFunction 227
- UCBioAPI_SC_ReadSerial 228
- UCBioAPI_SC_ReadBlock 229

- UCBioAPI_SC_WriteBlock 231
- UCBioAPI_SC_ReadSector 233
- UCBioAPI_SC_WriteSector 235
- UCBioAPI_SC_ReadSectorFieldContent 237
- UCBioAPI_SC_WriteSectorFieldContent 239
- UCBioAPI_SC_PreValue 241
- UCBioAPI_SC_ReadValue 243
- UCBioAPI_SC_IncrementValue 245
- UCBioAPI_SC_DecrementValue 247
- UCBioAPI_SC_WriteSectorTrailer 249
- UCBioAPI_SC_ReqA 251
- UCBioAPI_SC_WupA 252
- UCBioAPI_SC_Select 253
- UCBioAPI_SC_HaltA 254
- UCBioAPI_SC_Rats 255
- UCBioAPI_SC_PpsRequest 257
- UCBioAPI_SC_BlockFormat 259
- UCBioAPI_SC_Deselect 261
- UCBioAPI_SC_TypeA_ActiveState 262

6. API Reference for COM 263

6.1. UCBioBSP Object 263

6.1.1. Methods 263
- SetSkinResource 263
- SetCaptureQuality 263

6.1.2. Properties 264
- ErrorCode 264
- ErrorDescription 264
- Device 264
- Extraction 265
- Matching 265
- FPData 265
- FPImage 266
- FastSearch 266
- SmartCard 266
- CheckValidityModule 266
- MajorVersion 267
- MinorVersion 267
- BuildNumber 267
- MaxFingersForEnroll 267

- NecessaryEnrollNum 268
- SamplesPerFinger 268
- DefaultTimeout 268
- SecurityLevelForEnroll / SecurityLevelForVerify / SecurityLevelForIdentify 268
- WindowStyle 269
- WindowOption 270
- ParentWnd 270
- FingerWnd 271
- CaptionMsg 271
- CancelMsg 271
- FPForeColor / FPBackColor 272
- DisableFingerForEnroll 272

6.2. IDevice Interface 273

6.2.1. Methods 273
- Open 273
- Close 273
- Enumerate 274
- Adjust 274

6.2.2. Properties 275
- ErrorCode 275
- ErrorDescription 275
- EnumCount 275
- EnumDeviceID 275
- EnumDeviceNameID 276
- EnumInstance 276
- EnumDeviceName 276
- EnumDeviceDescription 277
- EnumDeviceDll / EnumDeviceSys 277
- EnumDeviceAutoOn 277
- EnumDeviceBrightness / EnumDeviceContrast / EnumDeviceGain 278
- OpenedDeviceID 278
- DeviceNameID / DeviceInstance 278
- DeviceID 279
- ImageWidth / ImageHeight 279
- Brightness / Contrast / Gain 279
- IsFingerExisted 279

6.3. IExtraction Interface 281

6.3.1. Methods 281
- Auto Detect 281
- Capture 281

- Enroll 282

6.3.2. Properties 284

- ErrorCode 284
- ErrorDescription 284
- FIR 284
- FIRLength 284
- TextFIR 285
- FIRFormat 285

6.4. IMatching Interface 287

6.4.1. Methods 287

- VerifyMatch 287
- Verify 287

6.4.2. Properties 289

- ErrorCode 289
- ErrorDescription 289
- MatchingResult 289
- IsPayloadExisted 289
- Payload 290
- PayloadLength 290
- TextPayload 290

6.5. IFPData Interface 292

6.5.1. Methods 292

- ClearFPData 292
- Export 292
- Import 294
- CreateTemplate 296

6.5.2. Properties 297

- ErrorCode 297
- ErrorDescription 297
- TotalFingerCount 297
- FingerID 297
- SampleNumber 298
- FPSampleData 298
- FPSampleDataLength 299
- FIR 299
- FIRLength 300
- TextFIR 300
- FIRFormat 301

6.6. IFPImage Interface 302

6.6.1. Methods 302
- Export 302
- ExportEx 302
- Save 303

6.6.2. Properties 305
- ErrorCode 305
- ErrorDescription 305
- TotalFingerCount 305
- FingerID 305
- SampleNumber 306
- ImageWidth / ImageHeight 306
- RawData 307
- AuditData 307
- AuditDataLength 308
- TextAuditData 308

6.7. IFastSearch Interface 309

6.7.1. Methods 309
- AddFIR 309
- RemoveFp 309
- RemoveUser 310
- ClearDB 311
- SaveDBToFile 311
- LoadDBFromFile 311
- IdentifyUser 312

6.7.2. Properties 313
- ErrorCode 313
- ErrorDescription 313
- FpCount 313
- FpInfo 313
- IsFpExisted 314
- AddedFpCount 314
- AddedFpInfo 315
- MatchedFpInfo 315
- MaxSearchTime 316
- UseGroupMatch 316
- MatchMethod 316

6.8. ITemplateInfo Interface 318

6.8.1. Properties 318
- UserID 318
- FingerID 318

- SampleNumber 318

6.9. ISmartCard Interface 319

6.9.1. Methods 319

- RFPowerOn 319
- RFPowerOff 320
- RFFunction 321
- ReadSerial 322
- ReadBlock 323
- WriteBlock 324
- ReadSector 325
- WriteSector 326
- ReadSectorFieldContent 327
- WriteSectorFieldContent 328
- PreValue 329
- ReadValue 330
- IncrementValue 331
- DecrementValue 332
- WriteSectorTrailer 333
- ReqA 334
- WupA 334
- Select 334
- HaltA 335
- Rats 336
- PpsRequest 337
- BlockFormat 338
- Deselect 339
- TypeA_ActiveState 340

6.9.2. Properties 341

- ErrorCode 341
- ErrorDescription 341
- LED 341
- AuthMode 341
- KeyA / KeyB 342
- ResultBuffer 342
- ResultLength 343
- Value 343
- Serial 344

7. Distribution Guide 345

7.1. Common Items 345
- Device driver 345
- UCDevice.dll 345
- UCBioBSP.dll 345
- Skin file 345

7.2. Development Using DLL 345

7.3. Development Using COM 346
- UCBioBSPCOM.dll 346

7.4. Development Using .NET 346
- .NET Framework v2.0 or higher 346
- UNIONCOMM.SDK.UCBioBSP.dll 346

7.5. Development over BioAPI Framework 346
- BioAPI Framework v2.0 or higher 346
- UCBioBSP.dll distribution and registration 346

Appendix A. How to enroll fingerprint properly 347

A.1. Proper Way to Enter Fingerprint 347

A.2. Improper Fingerprint Input Method 348

A.3. Procedures to Handle Authentication Failure 348

A.4. Fingerprint Registration, More Convenient with This Way 349

A.5. Considerations Depending on the Condition of User Fingerprint 349

1.Overview

UCBioBSP SDK is a high-level SDK created to develop applications using Union Community's fingerprint recognition devices and it supports SPI for BioAPI v2.0 proposed by the BioAPI Consortium. Also, it is a Software Development Kit that additionally provides extended API similar in form as API used in BioAPI.

Because UCBioBSP SDK provides all APIs related to fingerprint authentication and GUI (Graphical User Interface) for registration and authentication, developers can add fingerprint recognition functions to their products in development with minimal effort.
Since APIs for the smart card are provided, writing and reading the data that a user desires onto the smart card can be easily accomplished.

For easy and convenient use, this document provides detailed descriptions on each of APIs and examples on their use.

1.1. Special Features of SDK

UCBioBSP SDK has the following special features.

- **BioAPI v2.0 compatibility**
  Supporting the international standard API BioAPI v2.0 (ISO/IEC 19784-1:2005) for bio authentication

- **Various programming languages provided**
  Providing modules that can be used in various languages such as C, C++, Visual Basic, Delphi and etc. & also providing samples

- **Easy to use GUI provided**
  Customization for each user possible by providing GUI optimized to fingerprint and skin type UI

- **Multi fingerprint support**
  Management of 10 fingerprints for each user as integrated data possible

- **Powerful encryption provided**
  Data security using international standard encryption algorithm AES (Advanced Encryption Standard)

- **Self-protection function**
  Self-protection function available to prevent modification and forgery of module

- **Smart card support**
  Providing the function of reading and writing values onto smart card (Mifare, ISO14443-A)

1.2. Provided Module

The following modules are included in UCBioBSP SDK.

- **UCBioBSP.dll**
  As the core module of UCBioBSP SDK, it is considered to be the main module that can achieve all functions related to fingerprint authentication. This module must be included in the development process using UCBioBSP SDK.
  APIs that can be used in C/C++ are provided and SPI for BioAPI v2.0 is also included to be registered and used in the BioAPI framework environment. (Detailed information on BioAPI can be found at http://www.bioapi.org or obtained from BioAPI framework providers.)
  Relevant sample codes can be found in the DLL folder and BioAPI folder of the provided Sample folder.

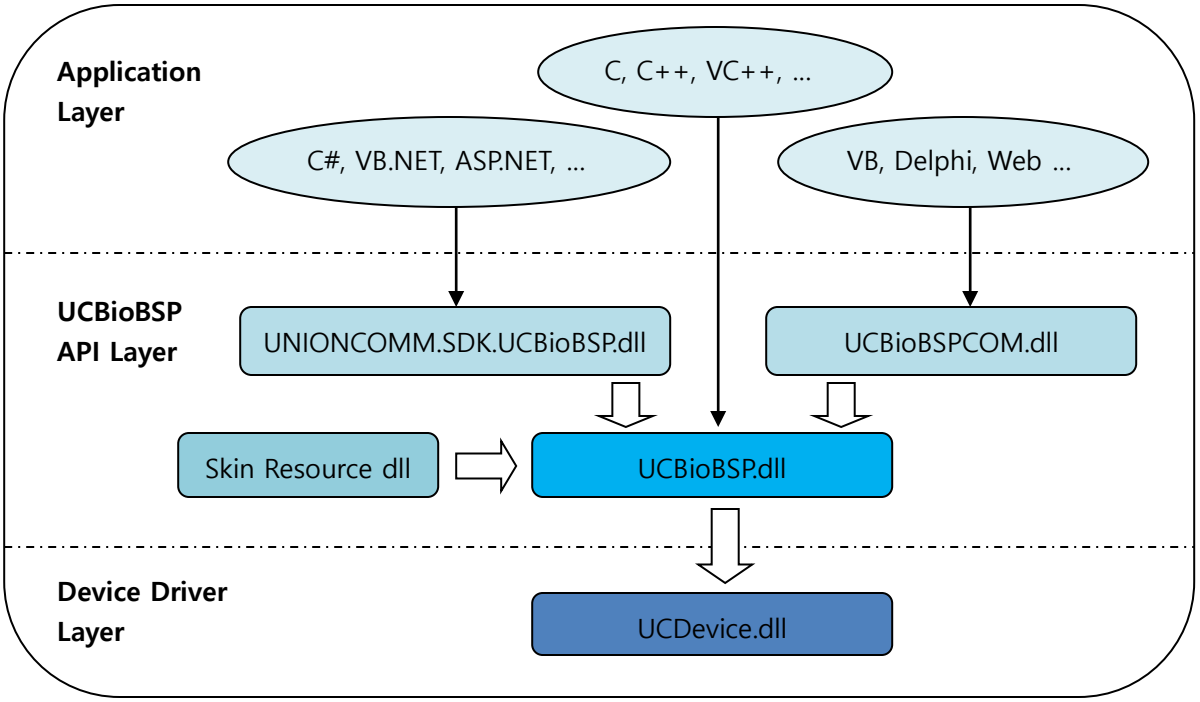
- **UCBioBSPCOM.dll**
  It is a COM (Component Object Model) module developed to support users of RAD (Rapid Application Development) tools such as Visual Basic and Delphi and the web development.
  Because UCBioBSPCOM.dll exists in a level above UCBioBSP.dll, it works only with the presence of UCBioBSP.dll. Also, it does not have all functions provided by UCBioBSP.dll but has some functions not provided by UCBioBSP.dll.
  Relevant sample codes can be found in the COM folder of the provided Sample folder.

- **UNIONCOMM.SDK.UCBioBSP.dll**
  It is a class library module for .NET that can be used in languages for the Microsoft .NET environment such as C#, VisualBasic.NET and ASP.NET.
  As in COM module, UNIONCOMM.SDK.UCBioBSP.dll exists in a level above UCBioBSP.dll and it works only with the presence of UCBioBSP.dll.
  Relevant sample codes can be found in the dotNET folder of the provided Sample folder.

- **Resource DLL**
  UCBioBSP SDK provides a method to use resource file for the skin produced outside. UCBioBSP.dll has the built-in skin in English by default. To use UI in a different language or form, external resource files can be read using skin related to API provided in UCBioBSP SDK.
  Currently, English and Korean skin resource DLLs are provided as default.
  Procedures to create the customized skin resource DLL are provided in a separate document.

1.3. Development Model

The structure of development model is given in the below figure.



1.4. Fingerprint Data Structure

In UCBioBSP SDK, the fingerprint data is represented in the form of FIR (Fingerprint Identification Record). As a data structure to represent fingerprint data of a person, a FIR can include fingerprint data of up to 10 fingers as well as image data.
FIR largely consists of three parts; format, header and data block.

Format	Header	Data Block
4Bytes	72Bytes	Variable size (=Header. DataLength)

1.4.1. Format

Format is a value that assigns the form of FIR, and this value can change the structures of header or FIR. Only the value of 1 that corresponds to UCBioAPI_FIR_FORMAT_STANDARD is currently available. When a new FIR structure is available in the future, this value is subject to change.

1.4.2. Header

Header consists of the following structures and its size is 72 bytes.

Header Length	Data Length	Version	Data Type	Purpose	Quality	Optional Data	Reserved
4Bytes	4Bytes	2Bytes	2Bytes	2Bytes	2Bytes	52Bytes	4Bytes

Length	UUIDInfo	PIN1	PIN2	Privilege	SiteID	IssueDate	ExpireDate	Reserved
4Bytes	20Bytes	4Bytes	4Bytes	4Bytes	4Bytes	4Bytes	4Bytes	4Bytes

- **Header Length**
  It has the length of header. This value is always 72.

- **Data Length**
  It has the length of data block, the fingerprint data. Fingerprint data is variable as is varies user by user.

- **Version**
  It shows the version information of FIR. This value is currently 1.

- **Data Type**
  It shows the fingerprint data type stored in FIR. FIR can have raw image data and special feature data of fingerprint and also indicates if data are encrypted.

- **Purpose**
  It shows a purpose of FIR data use. This value is different according to purpose of use, such as verify, identify, enroll, audit, etc.

- **Quality**
  It has the quality value of fingerprint data. This value ranges from 0 to 100. The higher the number is, the better quality of fingerprint data is.

- **Optional Data**
  It is a space in FIR to record additional informations. These values can be used for more refined authentication during a future fingerprint authentication. Information that can be added is provided below.

  ■ **Length**
  It has the length of optional data. This value is always 52.

  ■ **UUIDInfo**
  It can assign information on unique UUID (Universal Unique Identifier) value of FIR.
  It has a value of UCBioAPI_FIR_UUID_INFO structure.

  ■ **PIN1/PIN2**
  PIN (Personal Identification Number) information can be stored here.

  ■ **Privilege**
  FIR authority information can be assigned. The value ranges from 0 to 9. As the number gets bigger, a higher authority is required.

  ■ **Site ID**
  The ID of a specific site where FIR data are to be used can be assigned.

  ■ **Issue Date**
  The date of FIR data creation can be assigned.

  ■ **Expire Date**
  The expiration of date until when FIR data are valid can be assigned.

  ■ **Reserved**
  Reserved area

- **Reserved**
  Reserved area

1.4.3. Data Block

It is an area where fingerprint data are stored. It is an encrypted area in binary type. It can have special feature data or raw image data of fingerprint.

The size of fingerprint may vary depending on users and the size of data block may vary depending on the number of the registered fingers.

The size of data block can be found by referring to the data length of header.

1.5. Terminology Description

Terminologies used in this document are described here.

- **BSP**
  BSP is the abbreviation of Biometric Service Provider and it means service module for bio authentication.

- **Template / Sample**
  It means specializd fingerprint data for a single fingerprint.

- **Payload**
  It means an area inside FIR data to include specific information of a user. This kind of payload information is stored in encrypted state inside FIR and the value is returned only when authentication succeeds.

- **FIR**
  As the abbreviation of Fingerprint Identification Record, it is the user by user fingerprint data value. Template information or raw image information and payload information for a large number of fingers can be included in a single FIR.

- **FastSearch**
  It is the name of the high-speed authentication engine for 1:N authentication provided by UCBioBSP SDK.

- **BioAPI**
  A collection of Application Programming Interface for bio authentication designated by BioAPI Consortium is designated as the international standards (ISO/IEC 19784-1:2005). Detailed information can be obtained from http://www.bioapi.org.

2.Installation

2.1. System Requirements

The following system requirements are necessary to install UCBioBSP SDK.

- **OS**
  Supporting Windows 98SE/ME/2000/XP/2003/Vista, and other Windows OSs with USB support.

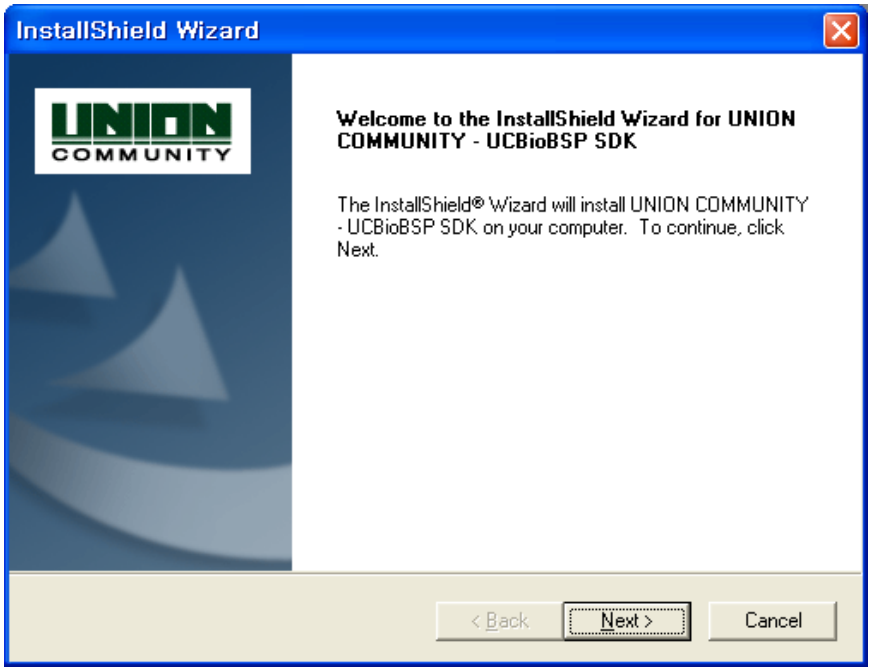
- **CPU**
  CPU above Pentium

- **Web environment**
  Server: IIS (Internet Information Server) 4.0 and higher
  Client: IE (Internet Explorer) 5.0 and higher

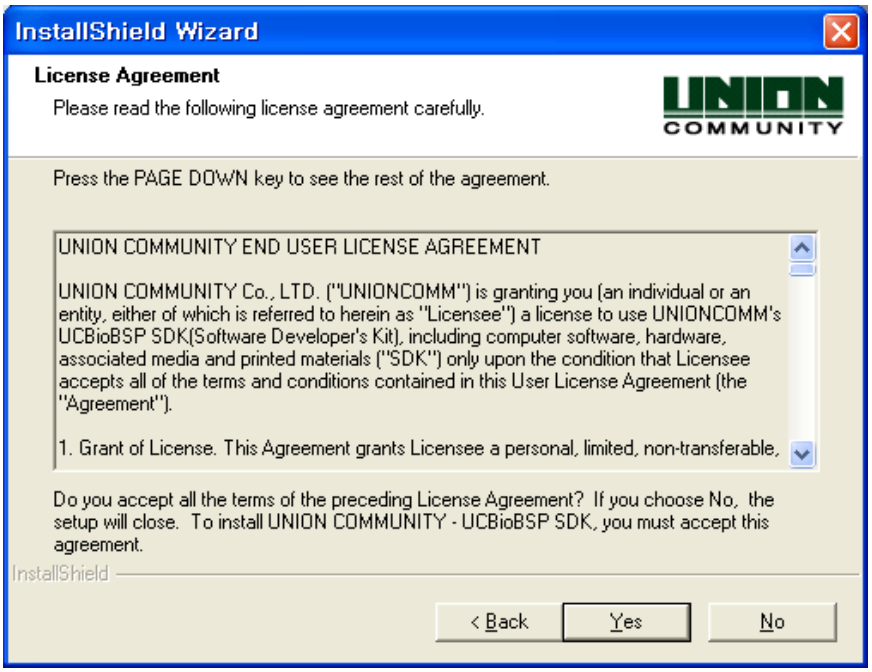
- **Device**
  Union Community's USB fingerprint recognition device for fingerprint acquisition.
  UCBioBSP SDK currently supports all fingerprint recognition devices for PC peripherals manufactured by Union Community. SDK will be provided for devices to be released in the future so that support is possible without modifying developed sources.
  Since UCBioBSP SDK does not include device drivers, a driver for each device must be separately installed.

2.2. Installing

1) Insert the installation CD into the drive.
2) Run Setup.exe file in the installation CD.
3) Click the Next button to start installation.



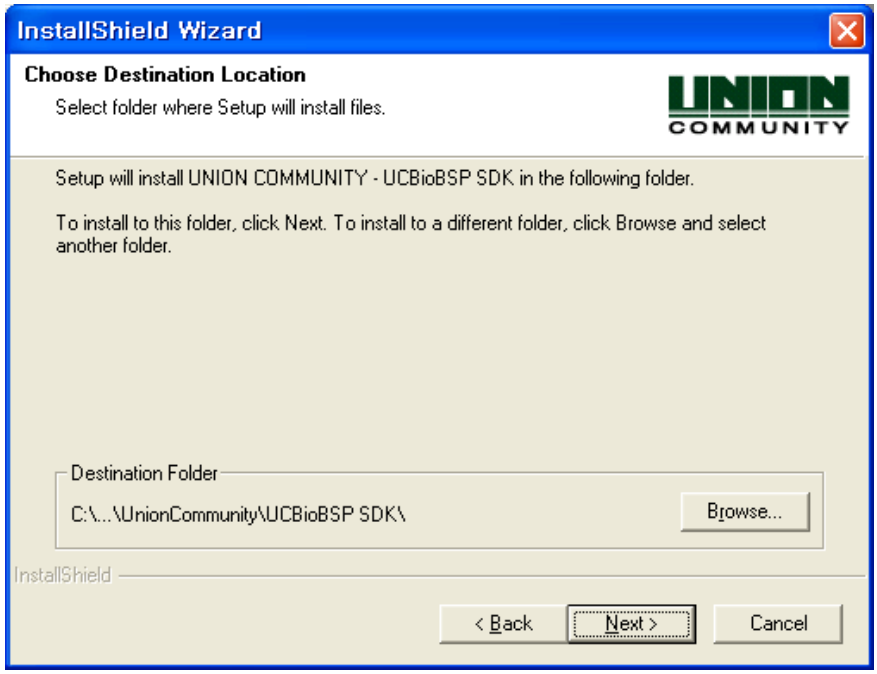
4) After checking the license agreement, click the Yes button if you agree with it.



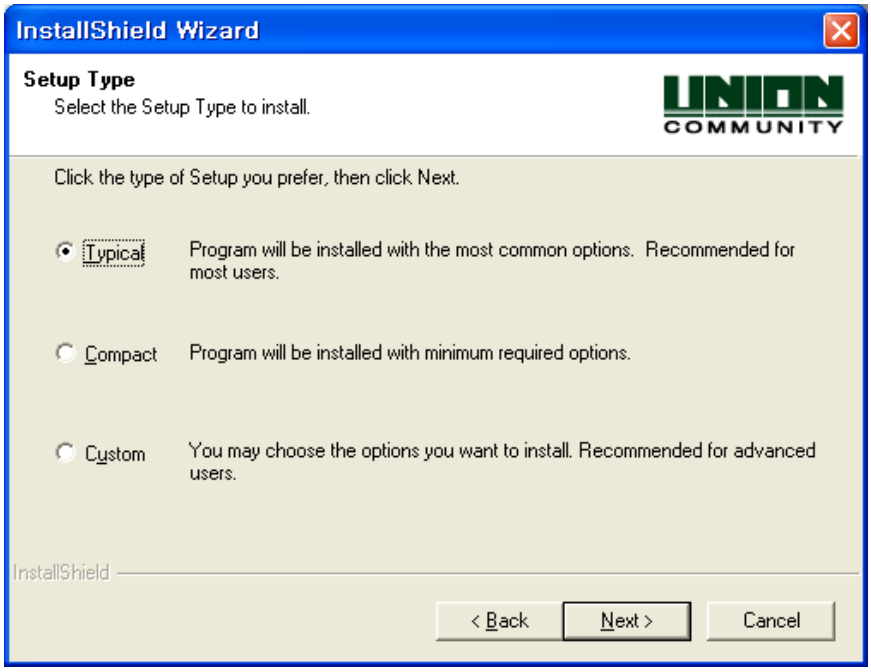
5) Enter user name, company name and serial number. The serial number can be obtained when purchasing SDK.



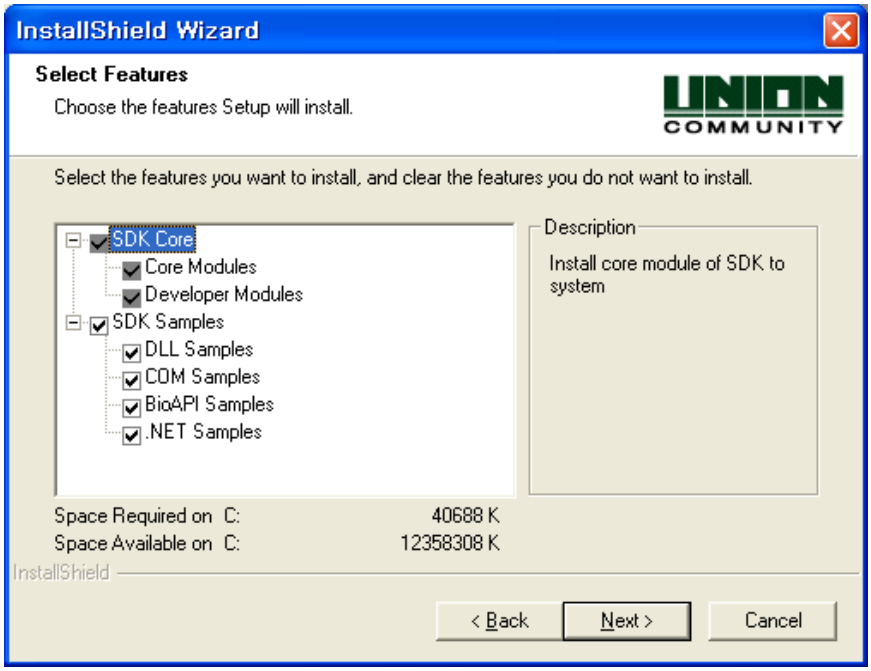
6) Designate the installation folder.



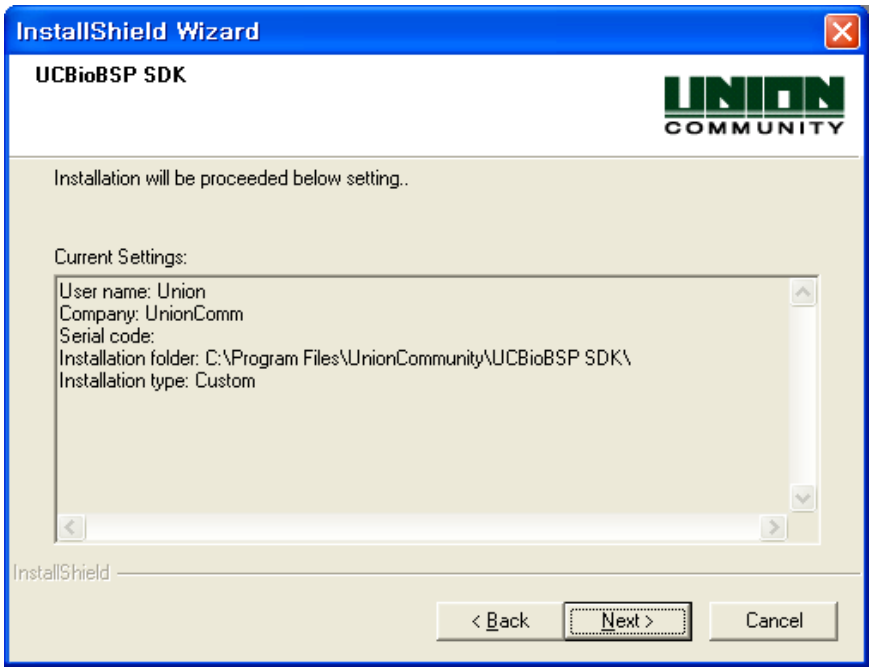
7) Installation type is designated. If Typical is selected, most components in SDK are installed as general installation. Compact installation excludes sample codes of SDK. To select components to be installed, select Custom installation.



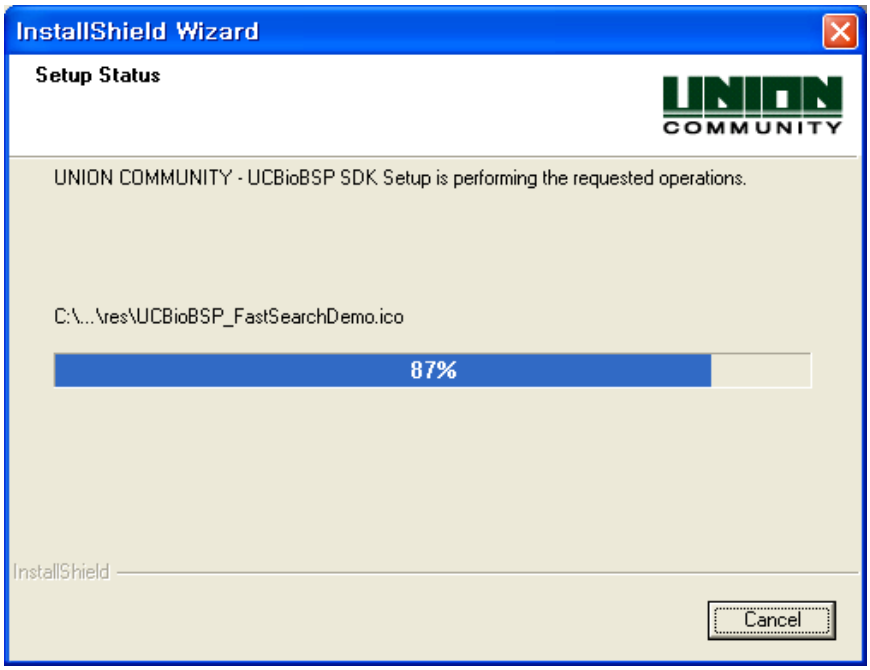
8) If Custom installation is selected, the following components can be selected and installed.



9) After checking all installation options, click the Next button to proceed.

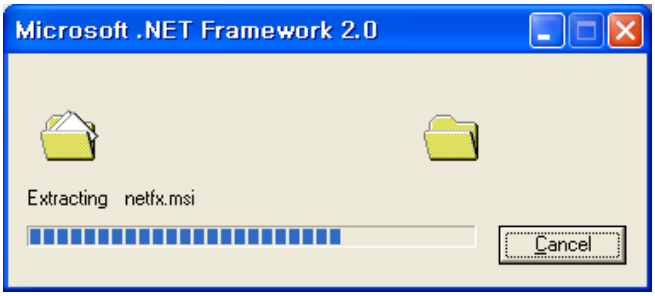


10) SDK installation is proceeding.

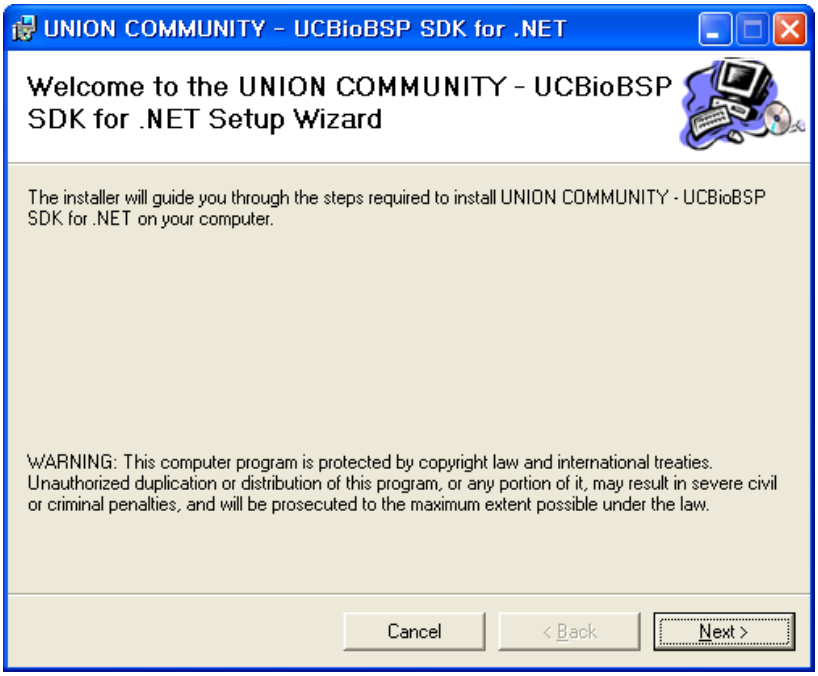


11) When file copying is completed, the prompt will ask you if module for .NET Framework should be installed. If .NET Framework is not used, click "No" not to install the class library for .NET. If necessary, module for .NET can be added later. But if development on .NET framework is required, click "Yes" to proceed with installation.

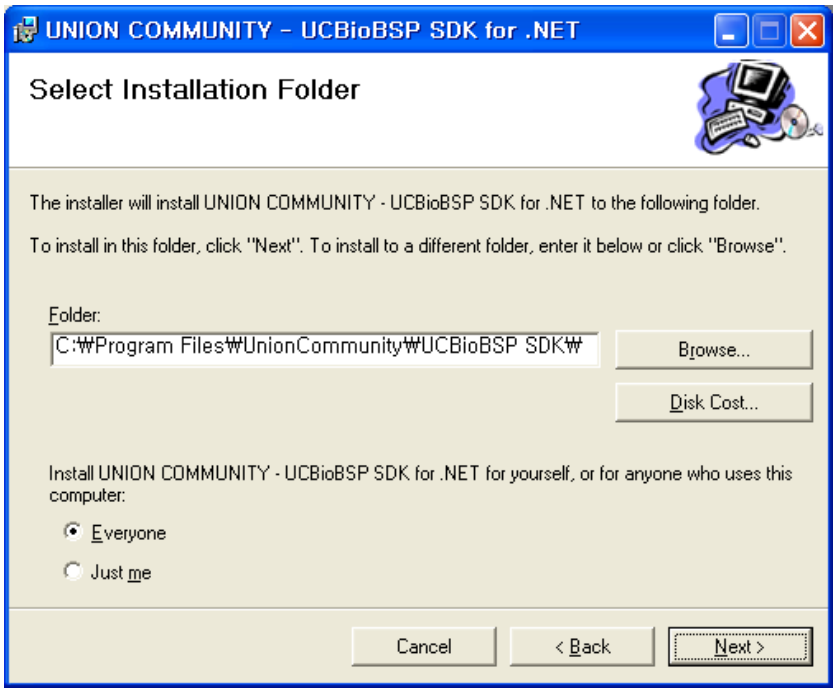
12) If "Yes" is selected, .NET Framework 2.0 is installed as shown in the following figure. If .NET Framework is already installed, this job can be canceled.



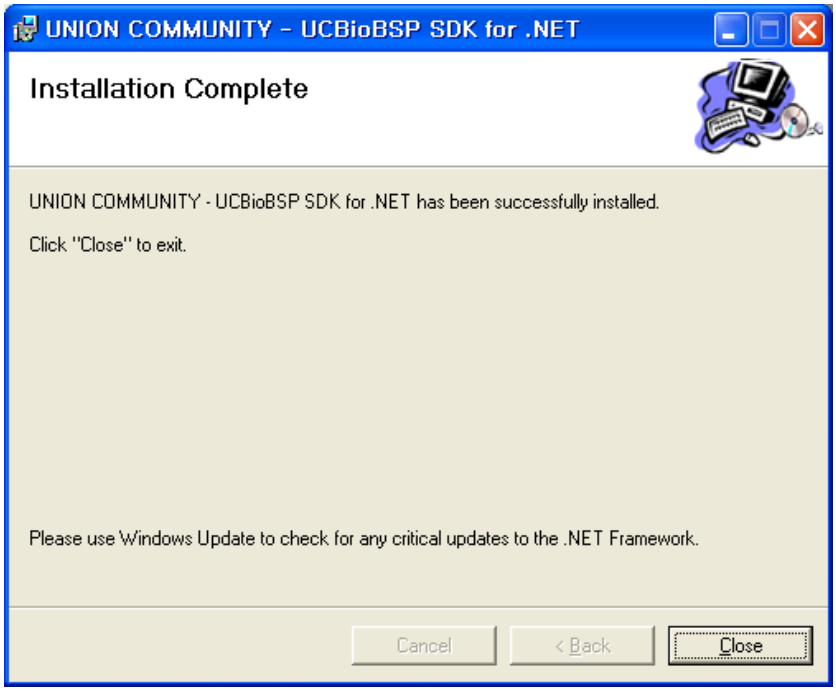
13) When .NET Framework installation is completed, the installation of class library for .NET proceeds. Click the Next button to start the installation.



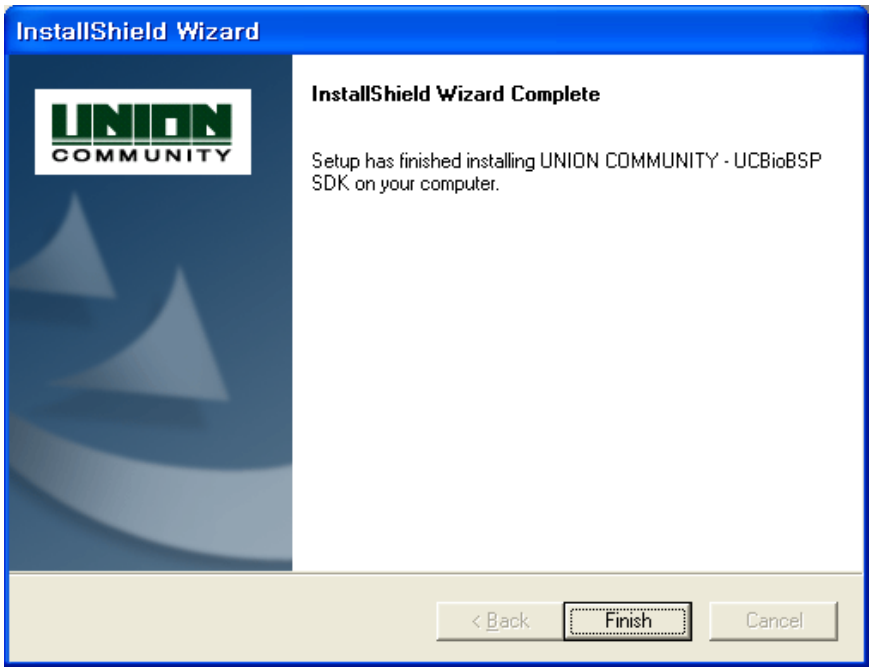
14) Designate the installation folder. Class library for .NET is installed at the Bin folder under the designated folder and it is automatically registered at GAC (Global Assembly Cache). Click the Next button to proceed with the installation.



15) When the installation of module for .NET is completed, click the Close button to finish installation program for .NET.



16) You will be returned to the original SDK installation screen to finish installation.



2.3. Description of Files and Folders to Be Installed

When installation is completed, the following files are installed in the system.

2.3.1. Windows System32 folder

Core modules for SDK use are installed. The files in below are installed.

■ **UCBioBSP.dll**

Core module of UCBioBSP SDK. It is responsible for performing all functions of SDK.
Since SPI (Service Provider Interface) for BioAPI is provided, it can be registered at BioAPI Framework.

■ **UCBioBSPCOM.dll**

COM module of UCBioBSP SDK.

2.3.2. GAC (Global Assembly Cache) folder

The below file is installed at the GAC folder where the class library for .NET Framework environment is installed. During the SDK installation, it is installed if library for .NET is installed.

■ **UNIONCOMM.SDK.UCBioBSP.dll**

Class library module for .NET.

2.3.3. (Installation folder)\Inc

Various header files required to develop under C/C++ using SDK are included in this folder.

■ **UCBioAPI.h**

If this file is included as the main header file of UCBioBSP SDK, UCBioAPI_Basic.h, UCBioAPI_Error.h, UCBioAPI_Type.h files are included internally and automatically.

■ **UCBioAPI_Basic.h**

The basic data types used in UCBioBSP SDK are defined.

■ **UCBioAPI_Error.h**

The error values used in UCBioBSP SDK are defined.

■ **UCBioAPI_Type.h**

The data types and structure informations used in UCBioBSP SDK are defined.

■ **UCBioAPI_Export.h**

The functions for FIR data conversion are defined.

■ **UCBioAPI_ExportType.h**
The data types and structure informations for FIR data conversion are defined.

■ **UCBioAPI_FastSearch.h**
The functions to use the search engine for 1:N matching are defined.

■ **UCBioAPI_FastSearchType.h**
The data types and structure informations to use the search engine for 1:N matching are defined.

■ **UCBioAPI_SmartCard.h**
The functions to use the smart card are defined.

■ **UCBioAPI_SmartCardType.h**
The data types and structure informations to use the smart card are defined.

2.3.4. (Installation folder)\Lib

Library files for link to develop under VC++ using SDK are included in this folder.

■ **UCBioBSP.lib**
Library file for link created for VC++. It is used to link UCBioBSP.dll statically under VC++.

2.3.5. (Installation folder)\Bin

Core files required in running SDK and sample execution files are included in this folder.

■ **UCBioBSP.dll / UCBioBSPCOM.dll**
Identical file to the one installed in the Windows system32 folder.

■ **UCBioBSPCOM.cab**
CAB file signed after compressing to enable a distribution of UCBioBSPCOM.dll over the Web.

■ **Demo application**
Some of demo programs to test functions of UCBioBSP SDK briefly are included in this folder. All demo program sources are provided at the Samples folder.

2.3.6. (Installation folder)\dotNET

Class library files for dotNET required in running SDK are included in this folder.

■ **UNIONCOMM.SDK.UCBioBSP.dll**
Class library module for .NET. Identical file to the one installed in GAC.

2.3.7. (Installation folder)\dotNet\Setup

Installation files to install class library for .NET at GAC are included in this folder.

■ **Setup.exe (UCBioBSP.NET_Setup.msi)**
Installation file of class library for .NET.

2.3.8. (Installation folder)\Samples

Sample source codes for each language are included separately folder by folder for distinction.

■ **BioAPI**
Sample codes for BioAPI running in BioAPI Framework are included. But, to run these samples, BioAPI Framework v2.0 must be installed and UCBioBSP.dll must be registered at Framework.

■ **COM**
Sample codes that can be developed using UCBioBSPCOM.dll are included.
1) VB6: Samples created for Visual Basic 6.0 are included.
2) ASP: Samples created for ASP (Active Server Page) running in IIS are included.

■ **DLL**
Sample codes that can be developed using UCBioBSP.dll are included.
1) VC6: Samples created for Visual C++ 6.0 are included.

■ **dotNET**
Samples codes that can be developed under Microsoft .Net environment using UNIONCOMM.SDK.UCBioBSP.dll are included.
1) C#: Samples created for VisualStudio.NET 2005, C# are included.

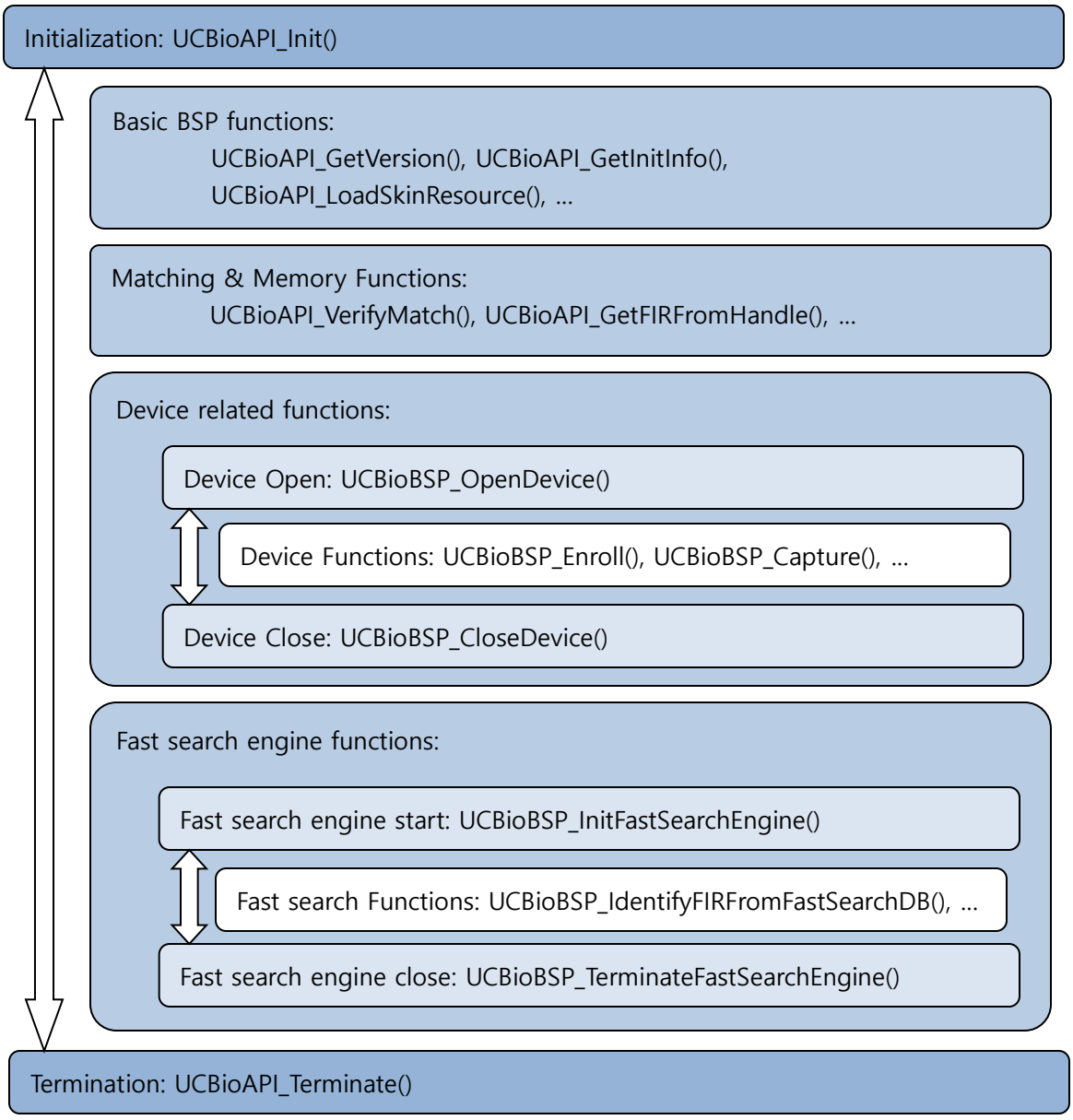
2.3.9. (Installation folder)\Skins

Languages by language skin resource files are included. Only English and Korean are currently included.

3.Programming by using DLL

This chapter describes how to program using DLL.
The language used in description is Visual C++ and other language users should make an appropriate modification for each language.

3.1. Function Call Structure



3.2. Initialization and Termination

How to initialize and terminate UCBioBSP SDK is described here.

3.2.1. Initializing

To use UCBioBSP SDK, the UCBioAPI_Init function is called first for initialization. UCBioAPI_Init function returns the Handle value of UCBioBSP SDK and this handle value is used in almost all functions provided by SDK.

■ **Example**

```
UCBioAPI_HANDLE m_hUCBioAPI;
…
UCBioAPI_RETURN err = UCBioAPI_Init(&m_hUCBioAPI);
if (err != UCBioAPIERROR_NONE) {
   // Failed to initialize UCBioBSP
} else {
   // Succeeded to initialize UCBioBSP
}
```

3.2.2. Terminating

To terminate a use of SDK, UCBioAPI_Terminate function must be called for termination. This process enables to release the memory used inside UCBioBSP SDK.

■ **Example**

```
UCBioAPI_HANDLE m_hUCBioAPI;
…
UCBioAPI_RETURN err = UCBioAPI_Terminate(m_hUCBioAPI);
if (err != UCBioAPIERROR_NONE) {
   // Failed to terminate UCBioBSP
} else {
   // Succeeded to terminate UCBioBSP
}
```

3.3. Basic Setting

Before using SDK following a successful initialization of UCBioBSP SDK, the basic setting can be obtained or new values can be assigned.

3.3.1. Obtaining SDK version

The BSP version of SDK currently in use can be obtained. This version consists of 4 steps.
(ex : 3.4.0.2)

■ **Example**

```
UCBioAPI_VERSION  ver;
memset(&ver, 0, sizeof(UCBioAPI_VERSION));

if (UCBioAPI_GetVersion(m_hUCBioAPI, &ver) == UCBioAPIERROR_NONE) {
   CString szVer;
   szVer.Format(_T("UCBioBSP Version : v%d.%4d.%d.%4d "),
        ver.Major1, ver.Major2, ver.Minor1, ver.Minor2);
   SetWindowText(szVer);
} else {
   // Failed to get version of BSP
}
```

3.3.2. Obtaining basic setting values & Setting up new values

The basic setting values of SDK can be obtained or new values can be set up.

■ **Example**

```
UCBioAPI_INIT_INFO_0 initInfo0;
memset(&initInfo0, 0, sizeof(UCBioAPI_INIT_INFO_0));

UCBioAPI_RETURN err = UCBioAPI_GetInitInfo(m_hUCBioAPI, 0, &initInfo0);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to get init information
} else {
   // Failed to get init information
}
```

■ **Example**

```
UCBioAPI_INIT_INFO_0 initInfo0;
memset(&initInfo0, 0, sizeof(UCBioAPI_INIT_INFO_0));

initInfo0.StructureType = 0;
```

```
initInfo0.MaxFingersForEnroll = m_nMaxFingersForEnroll;
initInfo0.NecessaryEnrollNum = m_nNecessaryEnrollNum;
initInfo0.SamplesPerFinger = m_nSamplesPerFinger;
initInfo0.DefaultTimeout = m_nDefaultTimeout;
initInfo0.SecurityLevelForEnroll = m_nSecuLevelForEnroll;
initInfo0.SecurityLevelForVerify = m_nSecuLevelForVerify;
initInfo0.SecurityLevelForIdentify = m_nSecuLevelForIdentify;

UCBioAPI_RETURN err = UCBioAPI_SetInitInfo(m_hUCBioAPI, 0, &initInfo0);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to set init information
} else {
   // Failed to set init information
}
```

UCBioAPI_INIT_INFO_0 structure represents the following values.

■ **StructureType**

It must have the value of 0.

■ **MaxFignersForEnroll**

The maximum number of fingers allowed for registration during fingerprint registration is designated. For example, assuming that this value is designated as 2, up to 2 fingers can be registered when registering fingerprint using UCBioAPI_Enroll function. The default for this value is 10.

■ **NecessaryEnrollNum**

The minimum number of fingers to be registered during fingerprint registration is designated. This value must be less than or equal to MaxFingersForEnroll value. For example, assuming that this value is designated as 2, more than 2 fingers have to be registered during fingerprint registration using UCBioAPI_Enroll function to complete registration process. The default for this value is 1.

■ **SamplesPerFinger**

During fingerprint registration, the number of samples to be stored for each finger is designated. It is fixed at 2 now and modification is not allowed.

■ **DefaultTimeout**

During fingerprint authentication and registration, the basic maximum time that the device operates to obtain fingerprint is designated in ms unit. Timeout can be separately designated later during function call and this value is used if

UCBioAPI_USE_DEFAULT_TIMEOUT(-1) is designated. The default for this value is 10000 (10 seconds).

■ **SecurityLevelForEnroll / SecurityLevelForVerify / SecurityLevelForIdentify**

Authentication security level used for fingerprint registration/authentication/1:N authentication can be set for each. Values in below are available.

```
#define UCBioAPI_FIR_SECURITY_LEVEL_LOWEST          (1)
#define UCBioAPI_FIR_SECURITY_LEVEL_LOWER           (2)
#define UCBioAPI_FIR_SECURITY_LEVEL_LOW             (3)
#define UCBioAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL    (4)
#define UCBioAPI_FIR_SECURITY_LEVEL_NORMAL          (5)
#define UCBioAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL    (6)
#define UCBioAPI_FIR_SECURITY_LEVEL_HIGH            (7)
#define UCBioAPI_FIR_SECURITY_LEVEL_HIGHER          (8)
#define UCBioAPI_FIR_SECURITY_LEVEL_HIGHEST         (9)
```

Enroll/Verify has 5 and Identify has 6 as default value.

■ **Reserved1 / Reserved2**

It is not used as reserved area. It has the value of 0.

3.4. Using Device

For fingerprint registration and acquisition, a process of opening the fingerprint recognition device is necessary to use it. Functions required to use devices installed in the PC are described here.

3.4.1. Obtaining device list

Using the UCBioAPI_EnumerateDevice function, information such as the entire list and the total number of devices currently connected to the PC can be obtained before the use of a device.

■ **Example**

```
UCBioAPI_UINT32            m_nNumDevice;
UCBioAPI_DEVICE_ID*        m_pDeviceID;
UCBioAPI_DEVICE_INFO_EX*   m_pDeviceInfoEx;
…

int nIndex;
UCBioAPI_RETURN err = UCBioAPI_EnumerateDevice(m_hUCBioAPI,
                          &m_nNumDevice, &m_pDeviceID,
                          &m_pDeviceInfoEx);
if (err) {
   // Failed to enumerate device!
   return;
}

if (m_nNumDevice == 0) {   // No device
    nIndex = m_comboDeviceList.AddString(_T("--NO DEVICE--"));
    m_comboDeviceList.SetItemData(nIndex, UCBioAPI_DEVICE_ID_NONE);
    return;
}

for (UCBioAPI_UINT32 i = 0 ; i < m_nNumDevice; i++) {
   nIndex = m_comboDeviceList.AddString(m_pDeviceInfoEx[i].Name);
   m_comboDeviceList.SetItemData(nIndex, m_pDeviceInfoEx[i].NameID);
}
```

If the UCBioAPI_EnumerateDevice function is used, the total number of devices connected to the system is passed back as the second argument m_nNumDevice and device ID is passed back in array form as the third argument m_pDeviceID. Lastly, more detailed information on the device is passed back in array form as the fourth argument m_pDeviceInfoEx.
m_pDeviceID and m_pDeviceInfoEx are pointer arrays to structures UCBioAPI_DEVICE_ID and UCBioAPI_DEVICE_INFO_EX, respectively. Since the memory for them is managed internally by

UCBioBSP SDK, an application does not have to allocate or release the memory.

3.4.2. Opening device

Before the use of a device, it must be opened using the UCBioAPI_OpenDevice function. To open a specific device, use an argument to assign ID to the device to be opened. To automatically open the device used most recently, assign UCBioAPI_DEVICE_ID_AUTO to device ID.

■ **Example**

```
int nDeviceID = UCBioAPI_DEVICE_ID_AUTO;
UCBioAPI_RETURN err = UCBioAPI_OpenDevice(m_hUCBioAPI, nDeviceID);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to open device
} else {
   // Failed to open device
}
```

Values that can be used as device ID are listed in below.

```
#define UCBioAPI_DEVICE_NAME_FOH01            (0x01)
#define UCBioAPI_DEVICE_NAME_FOM01            (0x02)
#define UCBioAPI_DEVICE_NAME_FOH03            (0x03)
#define UCBioAPI_DEVICE_NAME_HAM500           (0x04)
#define UCBioAPI_DEVICE_NAME_FOH01A           (0x05)
#define UCBioAPI_DEVICE_NAME_FOM01A           (0x06)
#define UCBioAPI_DEVICE_NAME_FPR02            (0x07)
#define UCBioAPI_DEVICE_NAME_FSH01RF          (0x08)
#define UCBioAPI_DEVICE_NAME_FOH01RF          (0x09)
#define UCBioAPI_DEVICE_NAME_FR100            (0x0a) // 10
#define UCBioAPI_DEVICE_NAME_FPR02LFD         (0x0b) // 11
#define UCBioAPI_DEVICE_NAME_FOH01RFL         (0x0c) // 12
#define UCBioAPI_DEVICE_NAME_FSH01SC          (0x0d) // 13
#define UCBioAPI_DEVICE_NAME_FPR02_V30        (0x0e) // 14
#define UCBioAPI_DEVICE_ID_AUTO               (0x00ff) // 255
```

3.4.3. Closing device

When finishing the use of a device, its use must be terminated using the UCBioAPI_CloseDevice function. Here, the device opened the first must be closed.
Also, to open other device, the device currently in use must be closed first to open a new device.

■ **Example**

```
int nDeviceID = UCBioAPI_DEVICE_ID_AUTO;
UCBioAPI_RETURN err = UCBioAPI_CloseDevice(m_hUCBioAPI, nDeviceID);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to close device
} else {
   // Failed to close device
}
```

3.4.4. Obtaining device information

To obtain information on the device currently opened, the UCBioAPI_GetDeviceInfo function can be used. Information such as the size of image supported by the device can be obtained. (Portions of unimportant codes are grayed out.)

■ **Example**

```
UCBioAPI_UINT32    m_nDeviceWidth, m_nDeviceHeight;
…
int nDeviceID = UCBioAPI_DEVICE_ID_AUTO;
UCBioAPI_RETURN err = UCBioAPI_OpenDevice(m_hUCBioAPI, nDeviceID);
if (err == UCBioAPIERROR_NONE) {
   UCBioAPI_DEVICE_INFO_0 deviceInfo0;
   memset(&deviceInfo0, 0, sizeof(UCBioAPI_DEVICE_INFO_0));
   err = UCBioAPI_GetDeviceInfo(m_hUCBioAPI, nDeviceID, 0,
                                &deviceInfo0);
   if (err == UCBioAPIERROR_NONE) {
      m_nDeviceWidth = deviceInfo0.ImageWidth;
      m_nDeviceHeight = deviceInfo0.ImageHeight;
   }
} else {
   // Failed to open device
}
```

3.5. Understanding FIR Data

To register and authenticate fingerprint, understanding fingerprint data FIR used by UCBioBSP SDK is essential. The structure of fingerprint data was described in chapter 1 and things to know to use FIR practically are described here.

3.5.1. The type of FIR

FIR can be classified into the following three types.

■ **FIR Handle**

What the user obtains when registering fingerprint or acquiring image using the API provided by UCBioBSP SDK is FIR Handle values rather than real FIR data. Since the memory for this FIR Handle is managed internally in BSP, real FIR data must be obtained from Handle to store fingerprint data registered by the use at DB or transmit them over the network. To obtain real FIR data, function such as UCBioAPI_GetFIRFromHandle is required. When the use Handle is finished, the memory must be released using the UCBioAPI_FreeFIRHandle function.

■ **FIR (Full FIR)**

It is the structure, which includes encrypted binary memory block to real fingerprint data, obtained from FIR Handle. To obtain this value, function such as UCBioAPI_GetFIRFromHandle can be used. Refer to chapter 1 for more detailed description of this structure.

■ **Text encoded FIR**

It is the structure, which includes encrypted memory block in character string type, to real fingerprint data obtained from FIR Handle. To obtain this value, function such as UCBioAPI_GetTextFIRFromHandle can be used. For the Unicode, a support for both ANSI character string and Unicode is possible.

3.5.2. The use of FIR

To use FIR in a function such as authentication, a structure called UCBioAPI_INPUT_FIR must be used. Since this structure includes union structure, it is a structure that can have all three types of FIR described above.

```
typedef struct ucbioapi_input_fir {
   UCBioAPI_INPUT_FIR_FORM        Form;
   union {
      UCBioAPI_FIR_HANDLE_PTR     FIRinBSP;
      UCBioAPI_VOID_PTR           FIR;
      UCBioAPI_FIR_TEXTENCODE_PTR TextFIR;
   } InputFIR;
```

```
} UCBioAPI_INPUT_FIR, *UCBioAPI_INPUT_FIR_PTR;
```

The use of each FIR according to its type is shown below. (Portions of unimportant codes are grayed out.)

■ **FIR Handle**

```
UCBioAPI_FIR_HANDLE hFIR;
UCBioAPI_RETURN err = UCBioAPI_Capture(m_hUCBioAPI,
                                       UCBioAPI_FIR_PURPOSE_VERIFY,
                                       &hFIR,
                                       UCBioAPI_USE_DEFAULT_TIMEOUT,
                                       NULL,
                                       NULL);
…
UCBioAPI_INPUT_FIR inputFIR;
inputFIR.Form = UCBioAPI_FIR_FORM_HANDLE;
inputFIR.InputFIR.FIRinBSP = &hFIR;
…
UCBioAPI_BOOL bResult;
err = UCBioAPI_Verify(m_hUCBioAPI, &inputFIR, &bResult, NULL,
                      UCBioAPI_USE_DEFAULT_TIMEOUT, NULL, NULL);
…
UCBioAPI_FreeFIRHandle(hFIR);
```

■ **FIR (Full FIR)**

```
UCBioAPI_FIR fullFIR;
UCBioAPI_GetFIRFromHandle(hFIR, &fullFIR);
…
UCBioAPI_INPUT_FIR inputFIR;
inputFIR.Form = UCBioAPI_FIR_FORM_FULLFIR;
inputFIR.InputFIR.FIR = &fullFIR;
…
UCBioAPI_BOOL bResult;
err = UCBioAPI_Verify(m_hUCBioAPI, &inputFIR, &bResult, NULL,
                      UCBioAPI_USE_DEFAULT_TIMEOUT, NULL, NULL);
…
UCBioAPI_FreeFIR(&fullFIR);
```

■ **Text encoded FIR**

```
UCBioAPI_FIR_TEXTENCODE textFIR;
UCBioAPI_GetTextFIRFromHandle(hFIR, &textFIR, UCBioAPI_FALSE);
```

```
…
UCBioAPI_INPUT_FIR inputFIR;
inputFIR.Form = UCBioAPI_FIR_FORM_TEXTENCODE;
inputFIR.InputFIR.TextFIR = &textFIR;
…
UCBioAPI_BOOL bResult;
err = UCBioAPI_Verify(m_hUCBioAPI, &inputFIR, &bResult, NULL,
                      UCBioAPI_USE_DEFAULT_TIMEOUT, NULL, NULL);
…
UCBioAPI_FreeTextFIR(&textFIR);
```

3.5.3. Releasing FIR memory

When the use of FIR is finished and it is no longer needed, memory must be released. The method to release memory according to the type of FIR is shown here.

■ **FIR Handle**

Memory is released using the UCBioAPI_FreeFIRHandle function.

■ **FIR (Full FIR)**

Memory is released using the UCBioAPI_FreeFIR function.

■ **Text encoded FIR**

Memory is released using the UCBioAPI_FreeTextFIR function.

For a sample code to release memory in each case, refer to the sample codes used to describe the use of FIR above.

3.5.4. Conversion of FIR

To store FIR or transmit it over the network, Handle must be converted to Full FIR or Text encoded FIR. As the pointer is included inside the structure for both cases, it is necessary to make appropriate conversion into stream type data for storage or transmission.

The next are examples of converting each data into stream type data according to the type of FIR.

■ **FIR (Full FIR)**

```
UCBioAPI_FIR fullFIR;
UCBioAPI_GetFIRFromHandle(hFIR, &fullFIR);
…
```

```
UINT nHeaderLength = sizeof(fullFIR.Format) + fullFIR.Header.Length;
UINT nStreamLength = nHeaderLength + fullFIR.Header.DataLength;
BYTE* pFIRStream = new BYTE [nStreamLength];
If (pFIRStream) {
   memset(pFIRStream, 0, nStreamLength);
   memcpy(pFIRStream, &fullFIR, nHeaderLength);
   memcpy(pFIRStream+nHeaderLength, fullFIR.Data, fullFIR.DataLength);
}
…
if (pFIRStream)
   delete[] pFIRStream;
```

■ **Text encoded FIR**

```
UCBioAPI_FIR_TEXTENCODE textFIR;
UCBioAPI_GetTextFIRFromHandle(hFIR, &textFIR, UCBioAPI_FALSE);
…
UINT nHeaderLength = sizeof(textFIR.IsWideChar);
UINT nStringLength = strlen(textFIR.TextFIR);
UINT nStreamLength = nHeaderLength + nStringLength + 1;
BYTE* pFIRStream = new BYTE [nStreamLength];
If (pFIRStream) {
   memset(pFIRStream, 0, nStreamLength);
   memcpy(pFIRStream, &textFIR, nHeaderLength);
   memcpy(pFIRStream+nHeaderLength, textFIR.TextFIR, nStringLength);
}
…
if (pFIRStream)
   delete[] pFIRStream;
```

3.6. Registering Fingerprint

To register fingerprint, a device must be opened first. In UCBioBSP SDK, registered fingerprint is obtained in the form of FIR Handle. As the memory of FIR Handle is managed internally by BSP, FIR management functions need to be used to obtain real fingerprint data from FIR Handle.

3.6.1. New fingerprint registration & Existing fingerprint modification

To register fingerprint, the UCBioAPI_Enroll function is used. The fingerprint registration function not only can make registration but also can modify already registered fingerprints. To modify a fingerprint, assign an existing fingerprint to the second argument. To register a new fingerprint, assign NULL to this value.

3.6.2. Payload designation

During fingerprint registration, Payload can be designated to insert specific data of a user after the encryption into the inside of registered fingerprint data. Payload registered in this way can be obtained only if fingerprint authentication is successful.

3.6.3. Auto Detect

You can use this function for automatically detect the fingerprint input

3.6.4. Example of use

In the next example, how to change to new Payload data while modifying the already registered data can be found. More detailed examples can be found in the UCBioBSP_Demo folder in the Samples folder in the installation folder after SDK installation.

■ **Example**

```
UCBioAPI_FIR_HANDLE       m_hEnrolledFIR
…

UCBioAPI_FIR_HANDLE hNewFIR;

UCBioAPI_INPUT_FIR inputFIR;
inputFIR.Form = UCBioAPI_FIR_FORM_HANDLE;
inputFIR.InputFIR.FIRinBSP = &m_hEnrolledFIR;  // Existing FIR

UCBioAPI_FIR_PAYLOAD payload;
payload.Length = (lstrlen(m_szPayload) + 1) * sizeof(TCHAR);
payload.Data = new UCBioAPI_UINT8 [payload.Length];
memset(payload.Data, 0, payload.Length);
lstrcpy((LPTSTR)payload.Data, m_szPayload);
```

```
UCBioAPI_RETURN err = UCBioAPI_Enroll(m_hUCBioAPI, &inputFIR, &hNewFIR,
                    &payload, UCBioAPI_USE_DEFAULT_TIMEOUT, NULL, NULL);

if (err == UCBioAPIERROR_NONE) {
   UCBioAPI_FreeFIRHandle(m_hEnrolledFIR);
   m_hEnrolledFIR = hNewFIR;      // Replace new FIR to existing FIR
}
if (payload.Data)
   delete[] payload.Data;

// Setting for Auto Detect(FALSE:Not Use, TRUE:AutoDetect)
UCBioAPI_RETURN err = UCBioAPI_SetAutoDetect(m_hUCBioAPI, TRUE);
```

3.7. Acquiring Fingerprint

To acquire fingerprint, a device must be opened first.
Fingerprint data are obtained through FIR Handle as in fingerprint registration but only one live fingerprint is inputted unlike the fingerprint registration. (But, if the purpose of acquisition is set to registration, it is processed in the same way as the fingerprint registration process.)
After comparing fingerprint data acquired in this way with previously registered fingerprint data, fingerprint authentication can be performed.

3.7.1. Fingerprint acquisition

To acquire fingerprint, the UCBioAPI_Capture function is used. The purpose of acquisition can be designated and values that can be used are shown here.

```
#define UCBioAPI_FIR_PURPOSE_VERIFY                          (0x01)
#define UCBioAPI_FIR_PURPOSE_IDENTIFY                        (0x02)
#define UCBioAPI_FIR_PURPOSE_ENROLL                          (0x03)
#define UCBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY    (0x04)
#define UCBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY (0x05)
#define UCBioAPI_FIR_PURPOSE_AUDIT                           (0x06)
#define UCBioAPI_FIR_PURPOSE_UPDATE                          (0x10)
```

If purpose is designated as registration, it has the same effect as calling the UCBioAPI_Enroll function. A designated purpose is used only as reference, and it does not have any effect on future authentication.

3.7.2. Example of use

■ **Example**

```
UCBioAPI_FIR_HANDLE hCapturedFIR
UCBioAPI_RETURN err = UCBioAPI_Capture(m_hUCBioAPI,
                                       UCBioAPI_FIR_PURPOSE_VERIFY,
                                       &hCapturedFIR,
                                       UCBioAPI_USE_DEFAULT_TIMEOUT,
                                       NULL, NULL);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to capture FIR
} else {
   // Failed to capture FIR
}
```

3.8. Authenticating Fingerprint

It is not necessary to have a device to authenticate a fingerprint but a device must be opened first for authentication with a live fingerprint.
Largely, the following two methods are used for fingerprint authentication.

3.8.1. Authenticating live fingerprint with registered fingerprint

It is a method of comparing already registered fingerprint data given as a input value with a fingerprint entered from the current fingerprint recognition device in real time. Therefore, a device must be opened during use.
The UCBioAPI_Verify function is used.

■ **Example**

```
UCBioAPI_FIR_HANDLE hEnrolledFIR;
UCBioAPI_RETURN err = UCBioAPI_Enroll(m_hUCBioAPI,
                                      NULL,
                                      &hEnrolledFIR,
                                      NULL,
                                      UCBioAPI_USE_DEFAULT_TIMEOUT,
                                      NULL,
                                      NULL);
…
UCBioAPI_INPUT_FIR inputFIR;
inputFIR.Form = UCBioAPI_FIR_FORM_HANDLE;
inputFIR.InputFIR.FIRinBSP = &hEnrolledFIR;
…
UCBioAPI_BOOL bResult;
err = UCBioAPI_Verify(m_hUCBioAPI, &inputFIR, &bResult, NULL,
                      UCBioAPI_USE_DEFAULT_TIMEOUT, NULL, NULL);

if (err == UCBioAPIERROR_NONE && bResult) {
   // Succeeded to verify
} else {
   // Failed to verify
}
…
```

3.8.2. Authenticating already acquired fingerprint with registered fingerprint

Already registered fingerprint data and already acquired fingerprint data are given as input values and this method compares these two data. Therefore, it works without a device. It is a

method used in authenticating fingerprint data transmitted to the server from the client. The UCBioAPI_VerifyMatch function is used.

■ **Example**

```
UCBioAPI_FIR_HANDLE hEnrolledFIR;
UCBioAPI_RETURN err = UCBioAPI_Enroll(m_hUCBioAPI,
                                      NULL,
                                      &hEnrolledFIR,
                                      NULL,
                                      UCBioAPI_USE_DEFAULT_TIMEOUT,
                                      NULL,
                                      NULL);
…
UCBioAPI_FIR_HANDLE hCapturedFIR
UCBioAPI_RETURN err = UCBioAPI_Capture(m_hUCBioAPI,
                                      UCBioAPI_FIR_PURPOSE_VERIFY,
                                      &hCapturedFIR,
                                      UCBioAPI_USE_DEFAULT_TIMEOUT,
                                      NULL, NULL);
…
UCBioAPI_INPUT_FIR inputEnrolledFIR;
inputEnrolledFIR.Form = UCBioAPI_FIR_FORM_HANDLE;
inputEnrolledFIR.InputFIR.FIRinBSP = &hEnrolledFIR;

UCBioAPI_INPUT_FIR inputCapturedFIR;
inputCapturedFIR.Form = UCBioAPI_FIR_FORM_HANDLE;
inputCapturedFIR.InputFIR.FIRinBSP = &hCapturedFIR;
…
UCBioAPI_BOOL bResult;
err = UCBioAPI_VerifyMatch(m_hUCBioAPI,
                           &inputEnrolledFIR, &inputCapturedFIR,
                           &bResult, NULL);

if (err == UCBioAPIERROR_NONE && bResult) {
   // Succeeded to verify
} else {
   // Failed to verify
}
…
```

3.8.3. Obtaining Payload data

When fingerprint authentication is successful, the Payload inserted during fingerprint registration can be obtained. Because Payload data can be used as any data, they can be used to obtain a specific fixed value during the user's authentication. The next example is a code that obtains Payload during authentication.

■ **Example**

```
…
UCBioAPI_BOOL bResult;
UCBioAPI_FIR_PAYLOAD payload;
err = UCBioAPI_VerifyMatch(m_hUCBioAPI,
                           &inputEnrolledFIR, &inputCapturedFIR,
                           &bResult, &payload);

if (err == UCBioAPIERROR_NONE && bResult) {
   // Succeeded to verify
   // Use payload data…
   …
   // Free payload data
   UCBioAPI_FreePayload(&payload);
} else {
   // Failed to verify
}
…
```

3.9. Using FastSearch (1:N Authentication)

UCBioBSP SDK provides FastSearch Engine for 1:N high-speed authentication. When authenticating a large number of users, the repeated 1:1 authentication processes are not expected to result the efficient authentication speed. Therefore, the authentication only for 1:N is required and the function of authenticating 1 user out of a large number of users through FastSearch Engine is provided. This chapter explains API for FastSearch. To use FastSearch, the UCBioAPI_FastSearch.h file needs to be included.

3.9.1. Initialization and Termination

To initialize FastSearch Engine, the UCBioAPI_FastSearchEngine function needs to be called.
After this function is called, it enters a state to use FastSearch Engine.
When all works are completed, FastSearch Engine needs to be terminated. To terminate it, the UCBioAPI_TerminateFastSearchEngine needs to be called.

■ **Example**

```
#include "UCBioAPI_FastSearch.h"
…
// Initialize FastSearch Engine
err = UCBioAPI_InitFastSearchEngine(m_hUCBioAPI);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to initialize
   …

   // Terminate FastSearch Engine
   UCBioAPI_TerminateFastSearchEngine(m_hUCBioAPI);
} else {
   // Failed to verify
}
```

3.9.2. Obtaining basic setting values & Setting new values

The basic setting value of FastSearch Engine can be obtained or a new value can be set.
The UCBioAPI_GetFastSearchInitInfo function is used to obtain the basic setting value and the UCBioAPI_SetFastSearchInitInfo function is used to change values. For detailed information on each setting value, refer to the function reference.

3.9.3. Creating DB

To perform 1:N authentication, a large number of DBs to perform authentication must be created on memory first. For high-speed authentication, the authentication is performed after creating

memory DB internally. Each FIR data needs to be combined into a single memory DB for the authentication to do that. The UCBioAPI_AddFIRToFastSearchDB function creates a single memory DB from entered FIR data or the FIR data registered at the user DB.
Also, FastSearch Engine does not use FIR as single data unit but uses a template unit as data unit internally. Therefore, if several templates are included inside FIR, they can be added to DB internally even though a single FIR is added to DB.
FIR data to be registered at DB and the user ID value for those data are passed as input values. If 1:N authentication is successful later, the user ID passed at this point can be obtained.

■ **Example**

```
…
err = UCBioAPI_AddFIRToFastSearchDB(m_hUCBioAPI, &inputEnrolledFIR,
                                    nUserID, NULL);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to add FIR
   …
} else {
   // Failed to add FIR
}
…
```

3.9.4. Memory DB management

Various functions shown here are provided to manage the memory DB created for authentication.

■ **UCBioAPI_RemoveFpFromFastSearchDB**

A specific fingerprint is deleted from the memory DB.

■ **UCBioAPI_RemoveUserFromFastSearchDB**

All fingerprints of a specific user are deleted from the memory DB. As a large number of fingerprints of a specific user may exist in the memory DB, this function is useful to delete them altogether.

■ **UCBioAPI_ClearFastSearchDB**

The entire memory DB is deleted.

■ **UCBioAPI_GetFpCountFromFastSearchDB**

The number of fingerprints in the memory DB is obtained.

■ **UCBioAPI_GetFpInfoFromFastSearchDB**

Fingerprint information at a specific location in the memory DB is obtained.

■ **UCBioAPI_CheckFpExistInFastSearchDB**
It examines if specific fingerprint information exists in the memory DB.

■ **UCBioAPI_SaveFastSearchDBToFile**
The entire memory DB is saved as a file. The DB saved in this way can be read using the LoadFastSearchDBFromFile function.

■ **UCBioAPI_LoadFastSearchDBFromFile**
The DB saved as a file is read back to the memory DB.
This method can create the memory DB much faster than building a new DB using the UCBioAPI_AddFIRToFastSearchDB function.

For more detailed information on these functions, refer to the function reference. Detailed examples of use can be found in the UCBioBSP_FastSearchDemo folder in the Samples folder after SDK installation.

3.9.5. Authenticating 1:N

To authenticate the fingerprint of a specific user in the created memory fingerprint DB, the UCBioAPI_IdentifyFIRFromFastSearchDB function is used.
A security level value for authentication can be set during authentication and the Callback function that can obtain information at every authentication can be registered and used.

■ **Example**

Callback Function portion

```
UCBioAPI_RETURN WINAPI MyFastSearchCallBack
             (UCBioAPI_FASTSEARCH_CALLBACK_PARAM_PTR_0 pCallbackParam0,
              UCBioAPI_VOID_PTR pUserParam)
{
   MyInfo* pInfo = (MyInfo*)pUserParam;
   …

   if (pInfo->m_bStopFlag)
      return UCBioAPI_FASTSEARCH_CALLBACK_STOP;
   else
      return UCBioAPI_FASTSEARCH_CALLBACK_OK;
}
```

Identify portion

```
…
```

```
UCBioAPI_FIR_HANDLE hFIR;
ret = UCBioAPI_Capture(m_hUCBioBSP, UCBioAPI_FIR_PURPOSE_IDENTIFY,
                       &hFIR, UCBioAPI_USE_DEFAULT_TIMEOUT, NULL, NULL);
if (ret == UCBioAPIERROR_NONE) {
   UCBioAPI_FASTSEARCH_FP_INFO infoFp;

   UCBioAPI_INPUT_FIR inputFIR;
   inputFIR.Form = UCBioAPI_FIR_FORM_HANDLE;
   inputFIR.InputFIR.FIRinBSP = &hFIR;

   UCBioAPI_FASTSEARCH_CALLBACK_INFO_0 callbackInfo0;
   callbackInfo0.CallBackType = 0;
   callbackInfo0.CallBackFunction = MyFastSearchCallBack;
   callbackInfo0.UserCallBackParam = &myInfo;

   ret = UCBioAPI_IdentifyFIRFromFastSearchDB(pDlg->m_hUCBioBSP,
                                &inputFIR, 5, &infoFp, &callbackInfo0);

   if (ret != UCBioAPIERROR_NONE) {
      if (ret == UCBioAPIERROR_FASTSEARCH_IDENTIFY_STOP) {
         // User stop!
      } else {
         // Failed to identify fingerprint data from DB!
      }
   } else {
      // Succeeded to identify
   }
   UCBioAPI_FreeFIRHandle(hFIR);
}
…
```

In the above example, whenever authentication occurs during performing Identify after registering the Callback function, the Callback function along with the fingerprint information currently in authentication is called. Using this Callback function, a user can display the current state of progress and terminate the authentication in the middle of process.
When authentication is successful, the authenticated fingerprint information can be obtained using the infoFp structure passed over as an argument. More detailed examples can be found in the UCBioBSP_FastSearchDemo folder in the Samples folder after SDK installation.

3.10. Converting FIR Data

As mentioned previously, FIR data are a collection of data consisting of a large number of template data. Therefore, a conversion function needs to be used to obtain each template data independently from FIR data or make one FIR data using a large number of template data.
Also for Audit FIR that contains image, raw images for each finger can be obtained using conversion functions.
To use functions introduced in this chapter, the UCBioAPI_Export.h file must be included.

3.10.1. Extracting template data from FIR data

To extract template data from FIR data, the UCBioAPI_FIRToTemplate function is used. This function obtains template data as well as diversed FIR information. The memory for UCBioAPI_EXPORT_DATA obtained this way must be released using the UCBioAPI_FreeExportData function.

■ **Example**

```
#include "UCBioAPI_Export.h"
…
UCBioAPI_EXPORT_DATA exportData;
err = UCBioAPI_FIRToTemplate(m_hUCBioAPI, &inputFIR, &exportData,
                             UCBioAPI_TEMPLATE_TYPE_SIZE400);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to export data
   …

   // Free export data
   UCBioAPI_FreeExportData(&exportData);
} else {
   // Failed to export data
}
```

3.10.2. Creating FIR Handle using template data

To create FIR Handle from template data, the next three functions can be used.

■ **UCBioAPI_TemplateToFIR**

Unlike FIR, template data in general have only fingerprints specialized information but do not have other information such as finger information. Therefore, when simply using template data only for authentication after simple conversion to FIR Handle, this function is used. In here, finger information or other related informations are not included internally in FIR.

■ **UCBioAPI_TemplateToFIREx**

It is nearly identical to the UCBioAPI_TemplateToFIR function and one FIR is created by designating several templates in array. The size of each template must be the same and the length is passed as an argument of the function.

```
UCBioAPI_UINT8 pMyTemplateData[400*2];
// Load template data to pMyTemplateData
…

UCBioAPI_FIR_HANDLE hNewFIR;
err = UCBioAPI_TemplateToFIREx(m_hUCBioAPI, pMyTemplateData,
                               400*2, 400,
                               UCBioAPI_TEMPLATE_TYPE_SIZE400,
                               UCBioAPI_FIR_PURPOSE_VERIFY, &hNewFIR);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to import data
   …
} else {
   // Failed to import data
}
```

■ **UCBioAPI_ImportDataToFIR**

This is the function used when creating FIR after building all fingerprint information using the UCBioAPI_EXPORT_DATA structure. Because accurate data values can be designated in this case, a desired FIR Handle can be accurately created.

```
…
int fn = 1, sn = 1;

UCBioAPI_EXPORT_DATA exportData;
memset(&exportData, 0, sizeof(UCBioAPI_EXPORT_DATA));

exportData.Length = sizeof(UCBioAPI_EXPORT_DATA);
exportData.TemplateType = UCBioAPI_TEMPLATE_TYPE_SIZE400;
exportData.FingerNum = fn;
exportData.DefaultFingerID = UCBioAPI_FINGER_ID_RIGHT_THUMB;
exportData.SamplesPerFinger = sn;
exportData.FingerInfo = new UCBioAPI_FINGER_BLOCK [fn];
exportData.FingerInfo[0].Length = sizeof(UCBioAPI_FINGER_BLOCK);
exportData.FingerInfo[0].FingerID = UCBioAPI_FINGER_ID_RIGHT_THUMB;
exportData.FingerInfo[0].TemplateInfo =new UCBioAPI_TEMPLATE_BLOCK[sn];
```

```
exportData.FingerInfo[0].TemplateInfo[0].Length = 400;
exportData.FingerInfo[0].TemplateInfo[0].Data =new UCBioAPI_UINT8[400];
memcpy(exportData.FingerInfo[0].TemplateInfo[0].Data,
                                                    pMyTemplateData, 400);

UCBioAPI_FIR_HANDLE hNewFIR;
err = UCBioAPI_ImportDataToFIR(m_hUCBioAPI, &exportData,
                               UCBioAPI_FIR_PURPOSE_VERIFY, &hNewFIR);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to import data
   …

   // Free export data
   if (exportData.FingerInfo[0].TemplateInfo)
      delete[]exportData.FingerInfo[0].TemplateInfo;
   if (exportData.FingerInfo)
      delete exportData.FingerInfo;
} else {
   // Failed to import data
}
```

3.10.3. Conversion between template data

Conversion between template data in various types is done using the UCBioAPI_ConvertTemplateData function. The memory for data converted in this way must be released using the UCBioAPI_FreeData function.

Values in below can be currently used for template types allowed for mutual conversion.

```
#define UCBioAPI_TEMPLATE_TYPE_SIZE400          (400)
#define UCBioAPI_TEMPLATE_TYPE_SIZE800          (800)
#define UCBioAPI_TEMPLATE_TYPE_SIZE320          (320)
#define UCBioAPI_TEMPLATE_TYPE_SIZE256          (256)
#define UCBioAPI_TEMPLATE_TYPE_FMR              (1)
```

3.10.4. Extracting raw image from Audit FIR data

Audit FIR is the FIR data that includes image information obtained as so-called Audit Data when using the UCBioAPI_Capture or UCBioAPI_Enroll function. This Audit FIR has the identical structure as the general FIRs but it includes an image internally unlike the general FIRs. To extract a raw image from this Audit FIR data, the UCBioAPI_AuditFIRToImage function is used.

The memory for UCBioAPI_EXPORT_AUDIT_DATA obtained this way must be released using the

UCBioAPI_FreeExportAuditData function.

■ **Example**

```
#include "UCBioAPI_Export.h"
…
UCBioAPI_EXPORT_AUDIT_DATA exportAuditData;
err = UCBioAPI_AuditFIRToImage(m_hUCBioAPI, &inputFIR,
                               &exportAuditData);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to export audit data
   …

   // Free export audit data
   UCBioAPI_FreeExportAuditData(&exportAuditData);
} else {
   // Failed to export audit data
}
```

3.10.5. Creating Audit FIR Handle using raw image

To create Audit FIR Handle using a raw image, the UCBioAPI_ImageToAuditFIR function is used.

3.11. Setting UI

The methods to organize user interface used in UCBioBSP SDK are described here.

3.11.1. Loading skin file

The UCBioBSP SDK skin type UI for the screen is in use for registration and authentication. Therefore, if UI in other languages or other forms of UI need to be used instead of the standard UI provided by UCBioBSP SDK, a user defined skin can be created and used. The function to read the skin DLL created at this point is the UCBioAPI_SetSkinResource function.
The UCBioBSP SDK has a currently built-in skin in English as default. To change a skin in Korean, follow the procedures shown here.

■ **Example**

```
err = UCBioAPI_SetSkinResource("UCBioBSPSkin_Kor.dll");
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to change to new skin
} else {
   // Failed to change to new skin
}
```

To create a user defined skin, contact our company for assistance.

3.11.2. Changing UI property

The UCBioBSP SDK provides the function that allows a user to change UI related property for use. This function can be used by passing the UCBioAPI_WINDOW_OPTION structure to the UCBioAPI_Enroll function, UCBioAPI_Capture function or UCBioAPI_Verify function as an argument.

■ **UCBioAPI_WINDOW_OPTION**

```
typedef struct ucbioapi_window_option {
   UCBioAPI_UINT32                Length;
   UCBioAPI_WINDOW_STYLE          WindowStyle;
   UCBioAPI_HWND                  ParentWnd;
   UCBioAPI_HWND                  FingerWnd;
   UCBioAPI_CALLBACK_INFO_0       CaptureCallBackInfo;
   UCBioAPI_CALLBACK_INFO_1       FinishCallBackInfo;
   UCBioAPI_CHAR_PTR              CaptionMsg;
   UCBioAPI_CHAR_PTR              CancelMsg;
   UCBioAPI_WINDOW_OPTION_PTR_2   Option2;
} UCBioAPI_WINDOW_OPTION, *UCBioAPI_WINDOW_OPTION_PTR;
```

```
typedef struct ucbioapi_window_option_2 {
   UCBioAPI_UINT8                FPForeColor[3];
   UCBioAPI_UINT8                FPBackColor[3];
   UCBioAPI_UINT8                DisableFingerForEnroll[10];
   UCBioAPI_UINT32               Reserved1[4];
   UCBioAPI_VOID_PTR             Reserved2;
} UCBioAPI_WINDOW_OPTION_2, *UCBioAPI_WINDOW_OPTION_PTR_2;
```

Descriptions on each structure member are shown here.

■ **Length**

The length of the structure. Currently, it is the same as the value of sizeof (UCBioAPI_WINDOW_OPTION).

■ **WindowStyle**

The type of displaying Window on the screen can be designated. It determines if a Window is launched as a pop-up type or only a fingerprint is displayed in an area of another Window. Other flag values can be designated and used. Allowed values are shown here.

```
#define UCBioAPI_WINDOW_STYLE_POPUP            (0)
#define UCBioAPI_WINDOW_STYLE_INVISIBLE        (1)

#define UCBioAPI_WINDOW_STYLE_NO_FPIMG         (0x00010000)
#define UCBioAPI_WINDOW_STYLE_NO_WELCOME       (0x00020000)
#define UCBioAPI_WINDOW_STYLE_NO_TOPMOST       (0x00040000)
```

As the below three flags can be used repeatedly, the designation can be made using the OR operator. For detailed descriptions on each flag, refer to the API reference.

■ **ParentWnd**

Designating Handle of parent Window

■ **FingerWnd**

When WindowStyle is set as UCBioAPI_WINDOW_STYLE_INVISIBLE, Handle of Window where fingerprint image is drawn is designated. If a value is designated, the fingerprint image is displayed in a designated Window and acquired later during fingerprint acquisition process, and therefore a user defined acquisition UI can be created.

■ **CaptureCallBackInfo**

A Callback function to be called whenever capture occurs during fingerprint acquisition is

designated.

■ **FinishCallBackInfo**
A Callback function to be called just before Window is closed after the completion of job is designated. Through the Callback designated here, some event information related to fingerprint registration can be obtained during fingerprint registration.

■ **CaptionMsg**
Contents to be displayed on the caption of the message box opened when the Cancel button is pressed during fingerprint registration are designated.

■ **CancelMsg**
Cancel announcement message contents of the message box opened when the Cancel button is pressed during fingerprint registration are designated.

■ **Option2**
In addition, other options can be given through the UCBioAPI_WINDOW_OPTION_2 function. Descriptions on the structure members are shown here. If any hasn't set, NULL is designated.

◆ **FPForeColor**
A color to be displayed for fingerprint on the screen can be designated.

◆ **FPBackColor**
When fingerprint is displayed on the screen, its background color can be designated.

◆ **DisableFingerForEnroll**
During fingerprint registration, fingers not allowed for registration can be designated.

Detailed examples of use for each member value can be found in the UCBioBSP_UIDemo in the Samples folder after UCBioBSP SDK installation.

3.11.3. Using Callback

Callback functions that can be designated to the UCBioAPI_WINDOW_OPTION are described.
The definition of Callback functions allowed for use is shown here.

```
typedef struct ucbioapi_callback_info_0 {
  UCBioAPI_UINT32               CallBackType;
  UCBioAPI_WINDOW_CALLBACK_0    CallBackFunction;
  UCBioAPI_VOID_PTR             UserCallBackParam;
} UCBioAPI_CALLBACK_INFO_0, *UCBioAPI_CALLBACK_INFO_PTR_0;
```

```
typedef struct ucbioapi_callback_info_1 {
   UCBioAPI_UINT32                CallBackType;
   UCBioAPI_WINDOW_CALLBACK_1     CallBackFunction;
   UCBioAPI_VOID_PTR              UserCallBackParam;
} UCBioAPI_CALLBACK_INFO_1, *UCBioAPI_CALLBACK_INFO_PTR_1;
```

■ **CallBackType**

A type of the CallBackFunction is set. If this value is 0, the pointer to the UCBioAPI_WINDOW_CALLBACK_PARAM_0 structure is passed back as the first argument of the Callback function. If this value is 1, the pointer to the UCBioAPI_WINDOW_-CALLBACK_PARAM_1 structure is passed back. Currently, only two types (0 and 1) are supported. Each of structures is shown here.

```
typedef struct ucbioapi_window_callback_param_0 {
   UCBioAPI_UINT32          dwQuality;
   UCBioAPI_UINT8*          lpImageBuf;
   UCBioAPI_UINT32          dwDeviceError;

   UCBioAPI_UINT32          dwReserved[8];
   UCBioAPI_VOID_PTR        lpReserved;
} UCBioAPI_WINDOW_CALLBACK_PARAM_0;

typedef struct ucbioapi_window_callback_param_1 {
   UCBioAPI_UINT32          dwResult;

   UCBioAPI_UINT32          dwStartTime;
   UCBioAPI_UINT32          dwCapTime;
   UCBioAPI_UINT32          dwEndTime;

   UCBioAPI_UINT32          Reserved[8];
   UCBioAPI_VOID_PTR        lpReserved;
} UCBioAPI_WINDOW_CALLBACK_PARAM_1;
```

The UCBioAPI_WINDOW_CALLBACK_PARAM_0 case is used for CaptureCallBackInfo of UCBioAPI_WINDOW_OPTION and UCBioAPI_WINDOW_CALLBACK_PARAM_1 case is used for FinishCallBackInfo. For detailed descriptions on each of members, refer to the API reference.

■ **CallBackFunction**

.The Callback function to be called is designated. The definition of each function according toCallBackType is shown here.

```
typedef UCBioAPI_RETURN (WINAPI* UCBioAPI_WINDOW_CALLBACK_0)
            (UCBioAPI_WINDOW_CALLBACK_PARAM_PTR_0, UCBioAPI_VOID_PTR);

typedef UCBioAPI_RETURN (WINAPI* UCBioAPI_WINDOW_CALLBACK_1)
            (UCBioAPI_WINDOW_CALLBACK_PARAM_PTR_1, UCBioAPI_VOID_PTR);
```

The value designated at UserCallBackParam that is designated next is passed back as the second argument.

■ **UserCallBackParam**

The user value passed back as the second argument of the Callback function is designated.

3.11.4. Example of use

An example of an actual use is shown here.

■ **Example**

```
…
UCBioAPI_WINDOW_OPTION winOption;
memset(&winOption, 0, sizeof(UCBioAPI_WINDOW_OPTION));

winOption.Length = sizeof(UCBioAPI_WINDOW_OPTION);
winOption.WindowStyle = UCBioAPI_WINDOW_STYLE_INVISIBLE;
winOption.ParentWnd = this;
winOption.FingerWnd = m_hwndFinger;
…

err = UCBioAPI_Capture(m_hUCBioAPI,
                       UCBioAPI_FIR_PURPOSE_VERIFY,
                       &hCapturedFIR,
                       UCBioAPI_USE_DEFAULT_TIMEOUT,
                       NULL,
                       &winOption);
…
```

More detailed examples of use can be found in the UCBioBSP_UIDemo in the Samples folder after UCBioBSP SDK installation.

3.12. Using the Smart Card

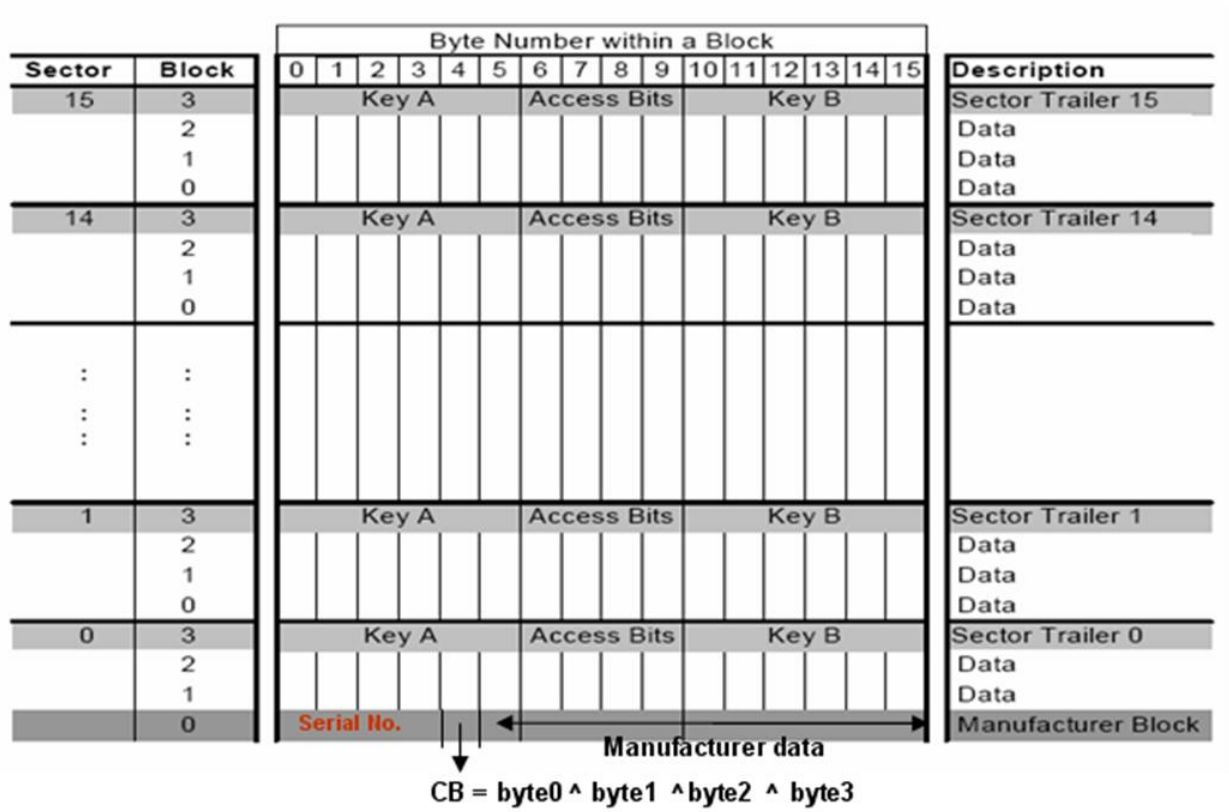
For a device supporting the smart card, UCBioBSP SDK provides a function to read and write onto the smart card. To use the smart card, first a device must be set in a state of use through the UCBioAPI_OpenDevice function and then functions related to the smart card are called.
How to use the smart card is described in this chapter. To use functions related to the smart card, the UCBioAPI_SmartCard.h file needs to be included.

- **Note – Among functions required to use the smart card, some of them may not be supported depending on the firmware version of a device.**

3.12.1. Outline of the smart card

All descriptions on the smart card can not be included in this document and detailed descriptions can be found through relavent documents. Here, brief descriptions on the internal structure and access rights of EEPROM of the Mifare card are given.

1) Structure of EEPROM



The above figure is that shows the EEPROM memory structure of a 1 Kbyte Mifare card.
As seen on the figure, EEPROM consists of 15 sectors (0~15) and each sector consists of 4 blocks

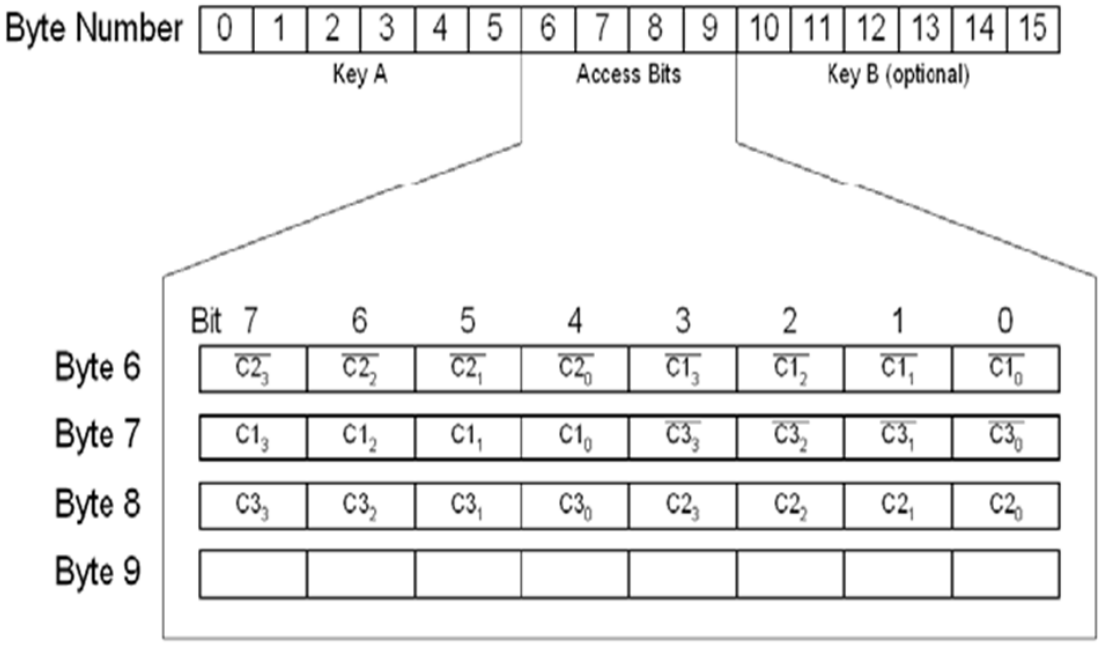
(0~3). That is, since the size of a block is 16 bytes, each sector has a space of 64 bytes and 16 of them make a space of 1 Kbyte in total.

```
16 Bytes X 4 Blocks X 16 Sectore = 1024 Bytes
```

The block 3 that is the fourth block of each sector has the key value for each sector and access bits for access rights, and it is called the sector trailer. General data can not be stored in this block and this value grants rights to read and write inside the sector.

2) Access Bits

4 bytes in the middle of the sector trailer are access bits that have the setting value for access rights of the sector. Values for access bits are shown in the next figure.



From values of each of bits, values marked with lower index denote the location of block. That is, $_3$ in $C1_3$ can be thought to represent the block 3. By this way, C1, C2 and C3 for block-by-block can be obtained. Values with line on top are parity values and they can be thought to be NOT value of each C1, C2 and C3.

Access rights to each of blocks can be designated using these C1, C2 and C3. The designation method is divided into the sector trailer part (block 3) and data block (block 0~2) and their meanings are shown in the next table.

■ **Access conditions for the sector trailer**

Access bits			Access condition for						Remark
			KEYA		Access bits		KEYB		
C1	C2	C3	read	write	read	write	read	write	
0	0	0	never	key A	key A	never	key A	key A	Key B may be read
0	1	0	never	never	key A	never	key A	never	Key B may be read
1	0	0	never	key B	key A|B	never	never	key B	
1	1	0	never	never	key A|B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	Key B may be read, transport configuration
0	1	1	never	key B	key A|B	key B	never	key B	
1	0	1	never	never	key A|B	key B	never	never	
1	1	1	never	never	key A|B	never	never	never	

■ **Access conditions for the data blocks**

Access bits			Access condition for				Application
C1	C2	C3	read	write	increment	decrement, transfer, restore	
0	0	0	key A|B[1]	key A|B1	key A|B1	key A|B1	transport configuration
0	1	0	key A|B[1]	never	never	never	read/write block
1	0	0	key A|B[1]	key B[1]	never	never	read/write block
1	1	0	key A|B[1]	key B[1]	key B[1]	key A|B[1]	value block
0	0	1	key A|B[1]	never	never	key A|B[1]	value block
0	1	1	key B[1]	key B[1]	never	never	read/write block
1	0	1	key B[1]	never	never	never	read/write block
1	1	1	never	never	never	never	read/write block
Access bits			Access condition for				Application

The meaning of each of values is shown here.

```
never : It can not be used with any method.
key A : It can be used if KeyA is known.
key B : It can be used if KeyB is known.
keyA|B : It can be used if KeyA or KeyB is known.
```

For more detailed information on access bits, refer to Mifare related documents.

3) Value Block

If a data block is changed to a value block, that block is changed to a block that can store a 4 bytes value as shown in the below figure. This changed value block can be used in a special purpose of increasing or decreasing the 4 bytes value written in the block by a fixed size. A value block is recorded over 3 times due to data preservation and security reason.

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Description	Value				$\overline{\text{Value}}$				Value				Adr	$\overline{\text{Adr}}$	Adr	$\overline{\text{Adr}}$

To be used as a value block, C1, C2 and C3 value of access bits need to be set to 1, 1 and 0 or 0, 0 and 1. UCBioBSP SDK provides a function called UCBioAPI_SD_PreValue for conversion to a value block.

3.12.2. Switching on/off RF power of smart card

To use the smart card, first the RF power of the smart card reader needs to be turned on. Through this way, reading and writing data from/to the smart card is possible. To turn on the RF power, the UCBioAPI_SC_RFPowerOn function is called.

To terminate the use of the smart card, the UCBioAPI_SC_RFPowerOff function is called to turn off the RF power of the reader.

The following can be used as an argument of the function. If UCBioAPI_SC_LED_TOGGLE is designated, a success and failure of the function are represented by red and blue in the LED of the smart card reader.

```
#define UCBioAPI_SC_LED_TOGGLE          (1)
#define UCBioAPI_SC_LED_NOT_TOGGLE      (0)
```

■ **Example - 1**

```
err = UCBioAPI_SC_RFPowerOn(m_hUCBioAPI, UCBioAPI_SC_LED_NOT_TOGGLE);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to RFPowerOn
} else {
   // Failed to RFPowerOn
}
```

■ **Example - 2**

```
err = UCBioAPI_SC_RFPowerOff(m_hUCBioAPI, UCBioAPI_SC_LED_NOT_TOGGLE);
```

```
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to RFPowerOff
} else {
   // Failed to RFPowerOff
}
```

3.12.3. Reading serial number of smart card

Every smart card has a unique serial number. To read this value, the UCBioAPI_ReadSerial function is used. Since this function has built-in UCBioAPI_RFPowerOn function, it is not necessary to turn on the RF power and know the key value.
It is convenient to use simply in reading the serial number of the smart card.

■ **Example**

```
BYTE pSerialBuffer[4] = { 0, 0, 0, 0 };
WORD nSerialBufferLen = 4;
err = UCBioAPI_SC_ReadSerial(m_hUCBioAPI,
                             pSerialBuffer,
                             &nSerialBufferLen,
                             UCBioAPI_SC_LED_TOGGLE);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to read serial
} else {
   // Failed to read serial
}
```

3.12.4. Reading & Writing block value

To read the value of each block value written on EEPROM of the smart card and read, the RF power must be turned on. To read a value, the UCBioAPI_SC_ReadBlock function is used. To write a value, the UCBioAPI_SC_WriteBlock function is used. The sector number and the block number where reading and writing are desired are designated as arguments, and key values appropriate for access rights of each block need to be passed.
An example of use is shown here.

■ **Example - 1**

```
BYTE pResultBuffer[16];
WORD nResultBufferLen = sizeof(pResultBuffer);
memset(pResultBuffer, 0, sizeof(pResultBuffer));

BYTE pKeyValue[6];
```

```
// Set key value to pKeyValue buffer

err = UCBioAPI_SC_ReadBlock(m_hUCBioAPI,
                            UCBioAPI_SC_USE_KEY_A,
                            nSectorNum, nBlockNum,
                            pKeyValue,
                            pResultBuffer,
                            &nResultBufferLen,
                            UCBioAPI_SC_LED_TOGGLE);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to read block
} else {
   // Failed to read block
}
```

■ **Example - 2**

```
BYTE pDataBuffer[16];
// Set data to pDataBuffer

BYTE pKeyValue[6];
// Set key value to pKeyValue buffer

err = UCBioAPI_SC_WriteBlock(m_hUCBioAPI,
                            UCBioAPI_SC_USE_KEY_A,
                            nSectorNum, nBlockNum,
                            pKeyValue,
                            pDataBuffer,
                            UCBioAPI_SC_LED_TOGGLE);
if (err == UCBioAPIERROR_NONE) {
   // Succeeded to write block
} else {
   // Failed to write block
}
```

For other functions related to the smart card, refer to the API reference.

4.Programming by using COM

This chapter describes how to program using COM.
Since description is made using Visual Basic 6.0, other programming language users need to modify them appropriately for each language before use.

4.1. Outline of COM Use

UCBioBSP SDK provides DLL for COM (Component Object Model) to support users of RAD (Rapid Application Development) tool such as Visual Basic or Delphi and web development. Since DLL for COM handles all the work internally using UCBioBSP.dll, UCBioBSP.dll must exist in the System32 folder of Windows before use.
COM does not support all functions provided by UCBioBSP.dll but convenience not provided by UCBioBSP.dll can be provided by using features of COM. Therefore, a user can develop using modules appropriate for needs.

4.1.1. Registration of COM

Because COM can be used after registration at the system registry, a process of registering the COM module at the system is required by entering the following line at the command line appeared after pressing [Windows key + R] button.

```
Regsvr32 UCBioBSPCOM.dll
```

After this process, the COM module can be used. If UCBioBSP SDK is installed, this process is implemented during the installation and therefore a separate registration process is not necessary in using 32bit COM. But in using 64bit COM, you should regist COM by yourself.

4.2. Initialization and Termination

How to initialize and terminate COM is described here.

4.2.1. Initializing

The COM module can be initialized by either declaring as a new object or using the CreateObject function. These two methods produce the same result.

■ **Example - 1**

```
Dim WithEvents objUCBioBSP As UCBioBSPCOMLib.UCBioBSP
…
' Create UCBioBSP object
Set objUCBioBSP = New UCBioBSPCOMLib.UCBioBSP
```

To use this method, "UNION COMMUNITY – UCBioBSP SDK v3.00 Type Library" needs to be included in usable reference items at "Reference" of Project Menu of Visual Basic before using the UCBioBSPCOM object.

■ **Example - 2**

```
Dim WithEvents objUCBioBSP As UCBioBSPCOMLib.UCBioBSP
…
' Create UCBioBSP object
Set objUCBioBSP = CreateObject("UCBioBSPCOMLib.UCBioBSP")
```

4.2.2. Terminating

Because an object whose use is finished is released automatically, it is not necessary to notify the termination of object use. However, it is necessary to clearly terminate it under a specific language or environment and the procedure is shown in the following example.

■ **Example**

```
' Free UCBioBSP object
Set objUCBioBSP = nothing
```

4.2.3. Lower interface declaration

7 lower interfaces exist for the COM module of UCBioBSP SDK. Lower interfaces can be obtained from the main object and functions are classified according to each of unique functions. Each of interfaces is shown here.

1) IDevice

Functions related to a device such as option setting of a device and opening/closing of a device are implemented.

```
Dim objDevice As IDevice          ' Declaration Device object
Set objDevice = objUCBioBSP.Device
```

2) IExtraction
Functions related to fingerprint data extraction such as acquisition and registration of fingerprint are implemented.

```
Dim objExtraction As IExtraction  ' Declaration Extraction object
Set objExtraction = objUCBioBSP.Extraction
```

3) IMatching
Functions related to fingerprint authentication are implemented.

```
Dim objMatching As IMatching      ' Declaration Matching object
Set objMatching = objUCBioBSP.Matching
```

4) IFPData
Functions related to data such as conversion of fingerprint data are implemented.

```
Dim objFPData As IFPData          ' Declaration FPData object
Set objFPData = objUCBioBSP.FPData
```

5) IFPImage
Functions related to extraction and storage of fingerprint image are implemented.

```
Dim objFPImage As IFPImage        ' Declaration FPImage object
Set objFPImage = objUCBioBSP.FPImage
```

6) IFastSearch
Functions related to 1:N engine FastSearch are implemented.

```
Dim objFastSearch As IFastSearch  ' Declaration FastSearch object
Set objFastSearch = objUCBioBSP.FastSearch
```

7) ISmartCard
Functions related to the smart card are implemented.

```
Dim objSmartCard As ISmartCard    ' Declaration SmartCard object
```

```
Set objSmartCard = objUCBioBSP.SmartCard
```

For more detailed descriptions on each of lower interfaces, refer to the COM reference.

4.3. Basic Setting

After successful COM initialization, the basic setting can be obtained or new values can be designated before the use of SDK.

4.3.1. Obtaining SDK version

The BSP version of SDK currently in use can be obtained. The version is consisted by 4 step.
MajorVersion return Major1 & Major2 and MinorVersion return Minor1 & Minor2.
(if version is 3.4.0.2 then MajorVersion return "3.4", MinorVersion return "0.2".)

■ **Example**

```
Set objUCBioBSP = New UCBioBSPCOMLib.UCBioBSP
' Check initialize object
If objUCBioBSP.ErrorCode = 0 Then
    ' Get UCBioBSP version
    Caption = objUCBioBSP.MajorVersion & "." & objUCBioBSP.MinorVersion
End If
```

4.3.2. Obtaining basic setting values & Setting new values

The basic setting values of SDK can be obtained or new values can be set.

■ **Example**

```
txtMaxFinger.Text = objUCBioBSP.MaxFingersForEnroll
txtNecessaryNum.Text = objUCBioBSP.NecessaryEnrollNum
txtSamplesPerFinger.Text = objUCBioBSP.SamplesPerFinger
txtDefaultTimeout.Text = objUCBioBSP.DefaultTimeout
txtEnrollSecuLevel.Text = objUCBioBSP.SecurityLevelForEnroll
txtVerifySecuLevel.Text = objUCBioBSP.SecurityLevelForVerify
txtIdentifySecuLevel.Text = objUCBioBSP.SecurityLevelForIdentify
```

■ **Example**

```
objUCBioBSP.MaxFingersForEnroll = txtMaxFinger.Text
objUCBioBSP.NecessaryEnrollNum = txtNecessaryNum.Text
objUCBioBSP.SamplesPerFinger = txtSamplesPerFinger.Text
objUCBioBSP.DefaultTimeout = txtDefaultTimeout.Text
objUCBioBSP.SecurityLevelForEnroll = txtEnrollSecuLevel.Text
objUCBioBSP.SecurityLevelForVerify = txtVerifySecuLevel.Text
objUCBioBSP.SecurityLevelForIdentify = txtIdentifySecuLevel.Text
```

As the property of UCBioBSP object that is the main object, the following values can be used.

■ **MaxFignersForEnroll**

When registering fingerprint, the maximum number of fingers that can be registered is designated. For example, if this value is set as 2, up to 2 fingers can be registered when registering fingerprint using the Enroll method of IExtraction.
The default for this value is 10.

■ **NecessaryEnrollNum**

When registering fingerprint, the minimum number of fingers that can be registered is designated. This value must be less than or equal to the MaxFingersForEnroll value.
For example, if this value is set as 2, at least 2 fingers need to be registered to complete a registration process when registering fingerprint using the Enroll method of IExtraction.
The default for this value is 1.

■ **SamplesPerFinger**

During fingerprint registration, the number of samples to be stored per finger is designated. Currently, it is fixed at 2 and modification is not allowed.

■ **DefaultTimeout**

During fingerprint authentication and registration, the basic maximum time during which a device operates to acquire a fingerprint is designated in ms unit. Timeout can be separately designated later during function call but this value is used if -1 is designated here.
The default for this value is 10000 (10 seconds).

■ **SecurityLevelForEnroll / SecurityLevelForVerify / SecurityLevelForIdentify**

Authentication security level to be used for fingerprint registration / authentication / 1:N authentication is set for each of them. This value can have values in below.

```
1 -  LOWEST
2 -  LOWER
3 -  LOW
4 -  BELOW_NORMAL
5 -  NORMAL
6 -  ABOVE_NORMAL
7 -  HIGH
8 -  HIGHER
9 -  HIGHEST
```

By default, Enroll/Verify has the value of 5 and Identify has 6.

4.4. Using Device

To register and acquire fingerprint, first a process of opening a fingerprint recognition device for use is necessary. Here, functions required to use devices connected to the PC are described.

4.4.1. Obtaining device list

Before the use of a device, information such as the entire list and the total number of devices currently connected to the PC can be obtained using the Enumerate method of IDevice.

■ **Example**

```
Dim i As Integer

' Enumerate Device
Call objDevice.Enumerate

If objUCBioBSP.ErrorCode <> 0 Then
    comboDevice.AddItem "Auto_Detect"
    For i = 0 To objDevice.EnumCount - 1
        nNameID = objDevice.EnumDeviceNameID(i)
        If nNameID <> 0 Then
            comboDevice.AddItem objDevice.EnumDeviceName(i)
        End If
    Next i
End If
```

After the Enumerate method of IDvevice is used, the total number of devices connected to the system is stored at the EnumCount property of IDevice and information on various devices is stored at several properties in array. After calling the Enumerate method, the types of property obtained are shown here.

■ **EnumDeviceID**

It includes the list of device ID. DeviceID is the addition value of DeviceNameID and DeviceInstance.

■ **EnumDeviceNameID**

It includes the ID list for device names.

■ **EnumDeviceInstance**

It includes the list for Instance numbers of devices. If several are connected to the same device, the number of Instance increases. Currently, only 0 is supported.

■ **EnumDeviceDescription**
It includes the list for descriptions of devices.

■ **EnumDeviceDll**
It includes the list for names of DLL files used by devices.

■ **EnumDeviceSys**
It includes the list of names of Sys files used by devices.

■ **EnumDeviceAutoOn**
It includes the list to show if devices support the AutoOn function.

■ **EnumDeviceBrightness / EnumDeviceContrast / EnumDeviceGain**
It includes the list of brightness value / contrast value / gain value.

4.4.2. Opening device

Before the use of a device, a device to be used must be opened using the Open method of IDevice. To open a specific device, set the ID of a device to be opened as an argument. To automatically open the device used most recently, set 255 to the device ID. (&HFF is used in Visual Basic.)

■ **Example**

```
Dim objUCBioBSP As UCBioBSPCOMLib.UCBioBSP
Dim objDevice As IDevice          ' Declaration Device object
…
' Create UCBioBSP object
Set objUCBioBSP = New UCBioBSPCOMLib.UCBioBSP
Set objDevice = objUCBioBSP.Device
…
' Open device
Call objDevice.Open(&HFF) ' &HFF = 0xFF = 255 = Auto

If objUCBioBSP.ErrorCode = 0 Then
   ' Succeeded to open device
Else
   ' Failed to open device
End If
```

Values that are allowed to be used as device ID are shown here.

```
1 -  FOH01
2 -  FOM01
3 -  FOH03
4 -  HAM500
5 -  FOH01A
6 -  FOM01A
7 -  FPR02
8 -  FSH01RF
9 -  FOH01RF
10 - FR100
11 - FPR02LFD
12 - FOH01RFL
13 - FSH01SC
14 - FPR02_V30
255- AUTO DETECT
```

4.4.3. Closing device

After the use of a device is finished, the device must be terminated using the Close method of IDevice. Here, the device opened the first must be closed.
Also, the device currently in use must be closed first to open a different device.

■ **Example**

```
' Close Device if before opened
Call objDevice.Close(objDevice.OpenedDeviceID)

If objUCBioBSP.ErrorCode = 0 Then
   ' Succeeded to close device
Else
   ' Failed to close device
End If
```

4.4.4. Obtaining device information

To obtain information on the device currently opened, values such as the size of image supported by the device can be obtained using various properties of IDevice.

■ **Example**

```
…
iDeviceID = &HFF
```

```
' Open device
Call objDevice.Open(iDeviceID)

If objUCBioBSP.ErrorCode = 0 Then
    txtWidth.Text = objDevice.ImageWidth(iDeviceID)
    txtHeight.Text = objDevice.ImageHeight(iDeviceID)
End If
```

4.5. Understanding FIR Data

To register and authenticate fingerprints, understanding on FIR that is fingerprint data used by UCBioBSP SDK is essential. The structure of fingerprint data was described in chapter 1, and things to know to use FIR in real COM are described here.

4.5.1. The type of FIR

FIR used in COM can be classified into the next two types.

■ **Binary FIR**

It is the value that includes encrypted binary memory block for fingerprint data. This value can be obtained through the FIR property of IExtraction.

■ **String FIR**

It is the encrypted character string type value for fingerprint data. This value can be obtained through the TextFIR property of IExtraction.

Two types of data produce the same result but String FIR is easier to use because it consists of character strings.

4.5.2. The use of FIR

To use FIR in functions such as authentication, just pass data in either Binary FIR or String FIR regardless as an argument of the method.

Method of use for each of them according to type is shown here.

■ **Example**

```
' Verify
If radioBinaryFIR.Value Then
    Call objMatching.Verify(binaryEnrolledFIR)  ' Verify with binary FIR
Else
    Call objMatching.Verify(szTextEnrolledFIR)  ' Verify with String FIR
End If
```

4.5.3. Releasing FIR memory

Because an object whose use is finished is released automatically in the COM module, it is not necessary to release the memory of the object.

4.6. Registering Fingerprint

To register a fingerprint, first a device must be opened.
To register a fingerprint in the COM module, the IExtraction interface must be used.
When a fingerprint is registered, fingerprint data are stored in the FIR or TextFIR property of IExtraction.

4.6.1. New fingerprint registration & Existing fingerprint modification

To register a fingerprint, the Enroll function of IExtraction is used. The fingerprint registration function can modify already registered fingerprints as well as register new fingerprints. To modify a fingerprint, designate an existing fingerprint at the second argument. To register a new fingerprint, set this value as NULL.

4.6.2. Payload designation

During fingerprint registration, Payload can be designated and then encrypted specific data of a user can be inserted into the inside of fingerprint data. Payload registered this way can be obtained only if authentication succeeds.

4.6.3. Auto Detect

You can use this function for automatically detect the fingerprint input

4.6.4. Example of use

The next example shows how to change to new Payload data while modifying already registered data. More detailed examples can be found in the UCBioBSPCOM_DemoVB in the Samples folder after SDK installation.

■ **Example**

```
Dim binaryEnrolledFIR() As Byte
Dim szTextEnrolledFIR As String
Dim szPayload As String

' Get Payload
szPayload = txtPayload.Text

' Enroll
Call objExtraction.Enroll(szPayload, szTextEnrolledFIR)

If objUCBioBSP.ErrorCode = 0 Then
   ' Get binary encoded FIR data
```

```
   binaryEnrolledFIR = objExtraction.FIR

   ' Get text encoded FIR data
   szTextEnrolledFIR = objExtraction.TextFIR

   labelStatus.Caption = "Succeeded to enroll."
Else
   labelStatus.Caption = "Failed to enroll."
End If


' Example for AutoDetect
Call objDevice.Open(&HFF) ' Device Open
Call objDevice.SetAutoDetect(0) ' Old type input
Call objDevice.SetAutoDetect(1) ' AutoDetect
```

4.7. Acquiring Fingerprint

To acquire a fingerprint, first a device must be opened.
To acquire a fingerprint in the COM module, the IExtraction interface must be used.
As in fingerprint registration, acquired fingerprint data are obtained in FIR. But unlike fingerprint registration, only one live fingerprint is inputted.
By comparing fingerprint data acquired this way and previously registered fingerprint data, fingerprint authentication is implemented.

4.7.1. Fingerprint acquisition

To acquire a fingerprint, the Capture method of IExtraction is used. A purpose of acquisition can be designated and values that are allowed to be used are shown here.

```
1 -  VERIFY
2 -  IDENTIFY
3 -  ENROLL
4 -  ENROLL_FOR_VERIFICATION_ONLY
5 -  ENROOL_FOR_IDENTIFICATION_ONLY
6 -  AUDIT
16 - UPDATE
```

If a purpose of registration is designated, it has the same effect as calling the Enroll method.
A designated purpose is used only for reference and it does not have any effect on a future authentication.

4.7.2. Example of use

■ **Example**

```
Dim binaryEnrolledFIR() As Byte
Dim szTextEnrolledFIR As String

' Capture
Call objExtraction.Capture(1)

If objUCBioBSP.ErrorCode = 0 Then
   ' Get binary encoded FIR data
   binaryEnrolledFIR = objExtraction.FIR

   ' Get text encoded FIR data
   szTextEnrolledFIR = objExtraction.TextFIR
```

```
   labelStatus.Caption = "Succeeded to capture."
Else
   labelStatus.Caption = "Failed to capture."
End If
```

4.8. Authenticating Fingerprint

To authenticate a fingerprint in the COM module, the IMatching interface must be used.
It is not essential to have a device to authenticate a fingerprint but a device must be opened first for authentication with a live fingerprint.
For fingerprint authentication, the next two methods are largely used.

4.8.1. Authentication live fingerprint wit registered fingerprint

This is a method of comparing a fingerprint entered in real time from the current fingerprint recognition device with already registered fingerprint data given as input value. Therefore, a device must be opened for use. To authenticate with this method, the Verify method of IMatching is used.

■ **Example**

```
' Verify
If radioBinaryFIR.Value Then
    Call objMatching.Verify(binaryEnrolledFIR)  ' Verify with binary FIR
Else
    Call objMatching.Verify(szTextEnrolledFIR)  ' Verify with String FIR
End If

If objUCBioBSP.ErrorCode <> 0 Then
    labStatus.Caption = "Verify Function Failed !"
    Exit Sub
End If

' Check result of verify
If objMatching.MatchingResult Then
    ' Check payload
    If objMatching.ExistPayload Then
        labelStatus.Caption = objMatching.TextPayload
    Else
        labelStatus.Caption = "Verify Succeeded!"
    End If
Else
    ' Show fail message.
    labStatus.Caption = "Verify Failed!"
End If
```

4.8.2. Authenticating already acquired fingerprint with registered fingerprint

Already registered fingerprint data and acquired fingerprint data are given as input values and two data are compared. Therefore, this method works without a device. This is a method used at the server for only authentication of fingerprint data transmitted from a client.
To authenticate with this method, the VerifyMatch method of IMatching is used.

■ **Example**

```
Dim szTextEnrolledFIR As String
Dim szTextCapturedFIR As String

' Enroll
Call objExtraction.Enroll(Null, Null)

If objUCBioBSP.ErrorCode = 0 Then
   ' Get text encoded FIR data
   szTextEnrolledFIR = objExtraction.TextFIR
End If

' Capture
Call objExtraction.Capture(1)

If objUCBioBSP.ErrorCode = 0 Then
   ' Get text encoded FIR data
   szTextCapturedFIR = objExtraction.TextFIR
End If

' VerifyMatch
Call objMatching.VerifyMatch(szCapturedFIR, szTextEnrolledFIR)

If objUCBioBSP.ErrorCode <> 0 Then
    labStatus.Caption = "VerifyMatch Function Failed !"
    Exit Sub
End If

' Check result of verify
If objMatching.MatchingResult Then
    ' Check payload
    If objMatching.ExistPayload Then
        labelStatus.Caption = objMatching.TextPayload
    Else
        labelStatus.Caption = "Verify Succeeded!"
    End If
```

```
Else
    ' Show fail message.
    labStatus.Caption = "Verify Failed!"
End If
```

4.8.3. Obtaining Payload data

When fingerprint authentication succeeds, Payload inserted during fingerprint registration can be obtained. Since Payload data can be used as any type of data, they can be used to obtain a specific fixed value during user authentication.

4.9. Using FastSearch (1:N Authentication)

UCBioBSP SDK provides FastSearch Engine for 1:N high-speed authentication. When authenticating a large number of users, an efficient authentication speed can not be expected simply through repeated 1:1 authentications. Therefore, authentication only for 1:N is required and a function to authenticate 1 user out of a large number of users is provided through FastSearch Engine. In the COM module, the IFastSearch needs to be used to use FastSearch Engine.

4.9.1. Initialization and Termination

In the COM module, special initialization or termination is not required to use FastSearch Engine. Simply obtaining the IFastSearch interface from the main object allows initialization to occur internally to get ready for use.

■ **Example**

```
Dim objUCBioBSP As UCBioBSPCOMLib.UCBioBSP
Dim objFastSearch As IFastSearch        ' Declaration FastSearch object
…
' Create UCBioBSP object
Set objUCBioBSP = New UCBioBSPCOMLib.UCBioBSP
Set objFastSearch = objUCBioBSP.FastSearch
…
```

4.9.2. Obtaining basic setting values & Setting up new values

Basic setting values of FastSearch Engine can be obtained or new values can be set.
Properties used as basic setting values are listed here.

■ **MaxSearchTime**

The maximum time to perform Identify is set. The unit is millisecond. Assume that the value of 10,000 is set. If authentication is not completed after performing Identify for 10 seconds, Identify terminates. If this value is set as 0, Identify is performed without time limit. The default value is 0.

■ **UseGroupMatch**

During performing authentication, if authentication is performed in group unit or not is determined. If this value is set as 0, authentication is performed according to the order in DB. If this value is set as 1, authentication in group unit is performed. The default value is 1.
It is recommended to set this value as 1 to perform authentication.

■ **MatchMethod**

A method to perform authentication is determined. If it is set as 0, the authentication level set up is used and authentication is immediately terminated when it goes over that level. If this value is set as 1, it uses the highest point authentication method and it searches for the value with the highest authentication level. The default value is 0. It is recommended to set this value as 0 to perform authentication.

4.9.3. Creating DB

To perform 1:N authentication, first a large number of DBs to perform authentication must be created on the memory. For high-speed authentication, authentication is performed by creating the memory DB internally. Combining each of FIR data into one memory DB for authentication is necessary to accomplish that. The AddFIR method of IFastSearch creates one memory DB from the entered FIR data or the FIR data registered at the user DB.

Also, FastSearch Engine internally does not use FIR as a single data unit but uses template unit as data unit. Therefore, if several templates are included inside FIR, internally several templates can be added to DB even if one FIR is added to DB.

The FIR data to be registered at DB and the user ID value for that data are passed as input values. When 1:N authentication succeeds later, the passed user ID value can be obtained.

■ **Example**

```
Dim nUserID As Long
Dim szFir As String
…
' Set User ID
nUserID = 1

' Get FIR data
Call objExtraction.Enroll(Null)
If objUCBioBSP.ErrorCode = 0 Then
    szFir = objExtraction.TextFIR

    ' Regist FIR to FastSearch DB
    Call objFastSearch.AddFIR(szFir, nUserID)
    If objUCBioBSP.ErrorCode = 0 Then
        MsgBox "Succeeded to add FIR to DB"
    Else
        MsgBox objUCBioBSP.ErrorDescription
    End If
Else
    MsgBox objUCBioBSP.ErrorDescription
End If
```

```
...
```

4.9.4. Memory DB management

To manage created memory DB for authentication, various methods as shown here are provided.

■ **RemoveFp**

One specific fingerprint is deleted from the memory DB.

■ **RemoveUser**

All fingerprints of a specific user are deleted from the memory DB. Since several fingerprint informations for a specific user can exist, this method is useful to delete them altogether.

■ **ClearDB**

The entire memory DB is deleted.

■ **FpCount**

The number of fingerprints in the memory DB is obtained.

■ **FpInfo**

Fingerprint information at a specific location in the memory DB is obtained.

■ **AddedFpCount**

The number of templates of the FIR added to DB just before is obtained.

■ **AddedFpInfo**

Template information of the FIR added to DB just before is obtained.

■ **IsFpExist**

It examines if specific fingerprint information exists in the memory DB.

■ **SaveDBToFile**

The entire memory DB is saved as a file. The DB saved this way can be read using the LoadDBFromFile method.

■ **LoadDBFromFile**

The DB saved as a file is loaded to the memory DB again.
This way can create the memory DB much faster than building a new DB using the AddFIR method.

4.9.5. Authenticating 1:N

To authenticate a fingerprint of a specific user in the created memory fingerprint DB, the IdentifyUser method of the IFastSearch interface is used.
During authentication, a security level value for authentication can be set.

■ **Example**

```
…
Dim i As Integer
Dim szTextFIR As String
Dim ListItem As ListItem
Dim objMatchedFpInfo As ITemplateInfo

' Set MaxTime for IdentifyUser
objFastSearch.MaxSearchTime = 5000  ' Set Maxtime to 5 seconds.

nUserID = 0
szTextFIR = ""

' Get FIR data
Call objDevice.Open(&HFF)    ' 0xFF : Auto detect to device ID
Call objExtraction.Capture(2)   ' 2 : Capture for identify purpose
If objUCBioBSP.ErrorCode = 0 Then
    szTextFIR = objExtraction.TextFIR

    ' Identify FIR to IndexSearch DB
    Call objFastSearch.IdentifyUser(szTextFIR, 5)

    If objUCBioBSP.ErrorCode = 0 Then
        Set objMatchedFpInfo = objFastSearch.MatchedFpInfo
        If objUCBioBSP.ErrorCode = 0 Then
            ' Add item to list of result
            Set ListItem = ListResult.ListItems.Add
            ListItem.Text = objMatchedFpInfo.UserID
            ListItem.SubItems(1) = objMatchedFpInfo.FingerID
            ListItem.SubItems(2) = objMatchedFpInfo.SampleNumber
            Set ListItem = Nothing
        End If
    Else
        MsgBox objUCBioBSP.ErrorDescription
    End If
Else
    MsgBox objUCBioBSP.ErrorDescription
```

```
End If

Call objDevice.Close(&HFF)   ' 0xFF : Auto detect to device ID
…
```

For detailed examples on FastSearch can be fond in the UCBioBSPCOM_FastSearchDemoVB in the Samples folder after SDK installation.

4.10. Converting FIR Data

As mentioned previously, FIR data are a collection of several template data. Therefore, to obtain each of template data from FIR data or create one FIR data with several template data, the conversion function is used.
Also, for Audit FIR that stores image, raw image for each of fingers can be obtained using provided conversion functions.
To use conversion related functions in the COM module, the IFPData and IFPImage interfaces need to be used.

4.10.1. Extracting template data from FIR data

To extract template data from FIR data, the Export method of IFPData is used.
When this method is called, not only template data but also various informations on FIR can be obtained.

■ **Example**

```
Dim biFIR() As Byte
Dim biTemplate() As Byte
Dim nTemplateType As Integer
nTemplateType = 400
…
objExtraction.Capture 1
…
' Get binary encoded FIR data
biFIR = objExtraction.FIR

' Export data
objFPData.Export biFIR, nTemplateType
If objUCBioBSP.ErrorCode = 0 Then
    ' Get template data
    nFingerID = objFPData.FingerID(0)
    biTemplate = objFPData.FPData(nFingerID, 0)
Else
    ' Failed to export data
End If
```

4.10.2. Creating FIR using template data

To create FIR from template data, the Import method of IFData is used. To create one FIR that includes several templates by calling the method several times, set the first argument as 0 and

call the Import method as many times as desired.
If this value is set as 1, the data imported previously are deleted and a new FIR is created.

■ **Example**

```
Dim biTemplate() As Byte
Dim nDataSize As Long
Dim nTemplateType As Integer
nTemplateType = 400
…
' Set template data to biTemplate buffer
…
' Import data
objFPData.Import 1, nFingerID, 1, nTemplateType, nDataSize, biTemplate
If objUCBioBSP.ErrorCode = 0 Then
    ' Verify
    objDevice.Open &HFF ' Auto
    objMatching.Verify objFPData.FIR
    objDevice.Close &HFF

    If objMatching.MatchingResult Then
        ' Succeeded to verify!
    Else
        ' Failed to verify!
    End If
Else
    ' Failed to import data
End If
```

4.10.3. Extracting raw image from Audit FIR data

Audit FIR is FIR data that include image information obtained when using the Capture or Enroll method of the IExtraction interface. This Audit FIR has the identical structure as a general FIR but a difference is that it includes image internally. To extract raw image from this Audit FIR data, the Export method of IFPImage is used.
Also, raw image obtained this way can be saved as image file in various formats using method such as the Save of IFPImage. Image formats allowed to be used are shown here.

```
1 -  RAW
2 -  BMP
3 -  JPG
4 -  WSQ
```

■ **Example**

```
Dim nImgType As Integer
nImgType = 2  ' BMP
…
objExtraction.Capture 1
…
' Export image data
objFPImage.Export
If objUCBioBSP.ErrorCode = 0 Then
    ' Save image data to file
    nFingerID = objFPData.FingerID(0)
    objFPImage.Save szFileName, nImgType, nFingerID
Else
    ' Failed to export data
End If
```

4.11. Setting UI

Methods to form user interfaces used in UCBioBSP SDK are described.
To set property related to UI in the COM module, the main interface of UCBioBSPCOM is used.

4.11.1. Loading skin file

UCBioBSP SDK uses skin type UI as the screen used for registration and authentication. Therefore, to use UI in other languages or other forms rather than the standard UI provided by UCBioBSP SDK, a user defined skin can be created and used. The method to load skin DLL created here is SetSkinResource.
Currently, UCBioBSP SDK has the built-in skin in English as default. To change this to a skin in Korean, follow the procedures below.

■ **Example**

```
' Set skin resource
Call objUCBioBSP.SetSkinResource(szSkinFileName)
If objUCBioBSP.ErrorCode = 0 Then
    // Succeeded to change to new skin
Else
   // Failed to change to new skin
End If
```

To create a user defined skin, contact our company.

4.11.2. Changing UI property

In UCBioBSP SDK, a function to allow a user to arbitrarily change and use UI related properties is available. COM provides setting values as property and each of these values is shown here.

■ **WindowStyle**

The type of displaying a Window on the screen is designated. It determines if a Window is launched as pop-up type or only a fingerprint is displayed in an area of another Window.

```
0 -  POPUP
1 -  INVISIBLE
```

■ **WindowOption**

The flag of the type how Window displays on the screen can be designated. Settings such as blocking display of a fingerprint on the screen or removing the Welcome page during fingerprint registration can be made. Since these three flags can be used repeatedly,

designation can be made using the OR operator.

```
0x00010000 – NO_FPIMG
0x00020000 – NO_WELCOME
0x00040000 – NO_TOPMOST
```

For detailed descriptions on each flag, refer to the API reference.

■ **ParentWnd**
Designating Handle of parent Window

■ **FingerWnd**
If WindowStyle is set as INVISIBLE(1), Handle of Window where fingerprint image is drawn is designated. If a value is designated here, fingerprint image is displayed on the designated Window during future fingerprint acquisition and therefore a user defined acquisition UI can be created.

■ **CaptionMsg**
When the Cancel button is pressed during fingerprint registration, contents to be displayed in the Caption of the message box are designated.

■ **CancelMsg**
When the Cancel button is pressed during fingerprint registration, the cancel announcement message contents in the message box are designated.

■ **FPForeColor**
.The color for a fingerprint to be displayed on the screen can be designated.

■ **FPBackColor**
When a fingerprint is displayed on the screen, background color can be designated.

■ **DisableFingerForEnroll**
During fingerprint registration, fingers to be prohibited from registration can be designated.

For detailed examples of each member value can be found in the UCBioBSPCOM_UIDemoVB in the Samples folder after SDK installation.

4.11.3. Using Callback Event Handler

When declaring objects for COM, an event can be received from a COM object if WithEvents is designated and declared as shown here.

```
' Declaration global variables
Dim WithEvents objUCBioBSP As UCBioBSPCOMLib.UCBioBSP
```

Events that can be received are the Capture event called every time Capture is implemented and the Enroll event called every time the Enroll method is implemented. Required values from each of Event Handler can be obtained.

■ **Example**

```
' Capture event handler
Private Sub objUCBioBSP_OnCaptureEvent(ByVal Quality As Long)
    …
End Sub

' Enroll event handler
Private Sub objUCBioBSP_OnEnrollEvent(ByVal EventID As Long)
    …
End Sub
```

For more detailed examples of use can be found in the UCBioBSP_UIDemo in the Samples folder after UCBioBSP SDK installation.

4.12. Using Smart Card

For devices that support the smart card, UCBioBSP SDK provides a function to read and write onto the smart card. To use the smart card, first a device needs to be ready for use through the IDevice.Open method to call functions related to the smart card.
This chapter describes how to use the smart card. To use functions related to the smart card, the ISmartCard interface is used.

- **Note – Functions to use the smart card may have some methods not supported depending on the firmware version of devices.**

For the outline of the smart card, refer to descriptions in section 3.12.1.

4.12.1. Initialization and Termination

To use the smart card in the COM module, special initialization or termination is not required. Getting the main object from the ISmartCard interface allows initialization to occur internally to get ready for use.

■ **Example**

```
Dim objUCBioBSP As UCBioBSPCOMLib.UCBioBSP
Dim objSmartCard As ISmartCard          ' Declaration SmartCard object
…
' Create UCBioBSP object
Set objUCBioBSP = New UCBioBSPCOMLib.UCBioBSP
Set objSmartCard = objUCBioBSP.SmartCard
…
```

4.12.2. Switching on/off RF power of smart card

To use the smart card, first the RF power of the smart card reader must be switched on. Then, reading and writing data onto the smart card is possible. To switch the RF power on, call the RFPowerOn method of ISmartCard.
To end the use of the smart card, call the RFPowerOff method and switch off the RF power of the reader.
If the LED property is set as 1, success/failure of the function is indicated as blue/red color in LED of the smart card reader.

```
0 -  LED_TOGGLE
1 -  LED_NOT_TOGGLE
```

■ **Example - 1**

```
objSmartCard.LED = 0    ' NOT Toggle LED
objSmartCard.RFPowerOn
If objUCBioBSP.ErrorCode = 0 Then
    ' Succeeded to RFPowerOn
Else
    ' Failed to RFPowerOn
End If
```

■ **Example - 2**

```
objSmartCard.LED = 0    ' NOT Toggle LED
objSmartCard.RFPowerOff
If objUCBioBSP.ErrorCode = 0 Then
    ' Succeeded to RFPowerOff
Else
    ' Failed to RFPowerOff
End If
```

4.12.3. Reading serial number of smart card

Every smart card has a unique serial number. To read this value, the ReadSerial method is used. Since this function has the built-in RFPowerOn function, it is not necessary to switch the RF power on separately and know the key value. It is convenient when simply reading the serial number. In the COM module, the serial number in either binary or long type can be obtained. The two values are identical.

■ **Example**

```
Dim serial() As Byte
Dim serialValue As Long

objSmartCard.ReadSerial

If objUCBioBSP.ErrorCode = 0 Then
    ' Succeeded to read serial
    serial = objSmartCard.ResultBuffer   ' binary type of serial
    serialValue = objSmartCard.serial    ' value type of serial
Else
    ' Failed to read serial
End If
```

4.12.4. Reading & Writing block value

To read and write the value of each block written on EEPROM of the smart card, the RF power must be switched on. The ReadBlock method is used to read the value and the WriteBlock method is used to write the value. Before calling the method, first the access rights of a block and the key value appropriate for the rights are designated as property. The sector number and block number where reading or writing is desired are designated as the arguments of each method.

Examples of use are shown here.

■ **Example - 1**

```
Dim blockData() As Byte
Dim key(6) As Byte
' Set key value to key array
…
objSmartCard.AuthMode = &H60      ' 0x60 - Use Key A
objSmartCard.KeyA = key

nSectorNum = 0
nBlockNum = 0

objSmartCard.ReadBlock nSectorNum, nBlockNum

If objUCBioBSP.ErrorCode = 0 Then
    ' Succeeded to read block
   blockData = objSmartCard.ResultBuffer
Else
    ' Failed to read block
End If
```

■ **Example - 2**

```
Dim blockData(16) As Byte
Dim key(6) As Byte

' Set block data to blockData array
' Set key value to key array
…
objSmartCard.AuthMode = &H60      ' 0x60 - Use Key A
objSmartCard.KeyA = key

nSectorNum = 0
```

```
nBlockNum = 0

objSmartCard.WriteBlock nSectorNum, nBlockNum, blockData
If objUCBioBSP.ErrorCode = 0 Then
    ' Succeeded to write block
Else
    ' Failed to write block
End If
```

For descriptions on other functions related to the smart card, refer to API references.

5.API Reference for DLL

This chapter describes types and APIs to use DLL module UCBioBSP.dll.

5.1. Type definitions

5.1.1. Basic types

Declaration is made in UCBioAPI_Basic.h and basic types are defined. Basic types are redefined for development independent of OS or CPU. Descriptions in below are based on development with C++ under standard Windows.

■ **UCBioAPI_SINT8 / UCBioAPI_SINT16 / UCBioAPI_SINT32**
Signed 1byte / 2bytes / 4bytes value

■ **UCBioAPI_UINT8 / UCBioAPI_UINT16 / UCBioAPI_UINT32 / UCBioAPI_UINT64**
Unsigned 1byte / 2bytes / 4bytes / 8bytes value

■ **UCBioAPI_SINT / UCBioAPI_UINT**
Int / Unsigned int value varying according to OS. It operates with 4 bytes for 32 bit OS and 8 bytes for 64 bit OS.

■ **UCBioAPI_VOID_PTR**
It means void*.

■ **UCBioAPI_BOOL**
It can have UCBioAPI_FALSE(0) / UCBioAPI_TRUE(1) value. It is handled in the same way as int.

■ **UCBioAPI_CHAR / UCBioAPI_CHAR_PTR**
It means char and char*. 1byte character and character string value.

■ **UCBioAPI_NULL**
It means NULL. It is defined as ((void*)0).

■ **UCBioAPI_HWND**
HWND value that means Handle of Window.

5.1.2. General types

Declaration is made in UCBioAPI_Type.h and general types are defined.

■ **UCBioAPI_FIR_VERSION**

Prototype:

```
typedef UCBioAPI_UINT16  UCBioAPI_FIR_VERSION;
```

Description:

It represents FIR data version number.

■ **UCBioAPI_VERSION**

Prototype:

```
typedef struct ucbioapi_version {
   UCBioAPI_UINT16     Major1;
   UCBioAPI_UINT16     Major2;
   UCBioAPI_UINT16     Minor1;
   UCBioAPI_UINT16     Minor2;
} UCBioAPI_VERSION, *UCBioAPI_VERSION_PTR;
```

Description:

The structure that includes BSP version number. If version is "v3.4.0.2" then version is stored in Major1=3, Major2 = 4, Minor1 = 0, Minor2 = 2.

■ **UCBioAPI_FIR_DATA_TYPE**

Prototype:

```
typedef UCBioAPI_UINT16   UCBioAPI_FIR_DATA_TYPE;
```

Description:

It represents the data type of FIR. Allowed values are shown below. Each of these values can be designated repeatedly through the OR operator.

```
#define UCBioAPI_FIR_DATA_TYPE_RAW           (0x00)
#define UCBioAPI_FIR_DATA_TYPE_INTERMEDIATE  (0x01)
#define UCBioAPI_FIR_DATA_TYPE_PROCESSED     (0x02)
#define UCBioAPI_FIR_DATA_TYPE_ENCRYPTED     (0x10)
```

If FIR is created, it has generally the value of (0x12).

■ **UCBioAPI_FIR_PURPOSE**

Prototype:

```
typedef UCBioAPI_UINT16   UCBioAPI_FIR_PURPOSE;
```

Description:

It represents the data purpose of FIR. Allowed values are shown below. This value is used only for the reference of FIR data and it does not have any effect on authentication.

```
#define UCBioAPI_FIR_PURPOSE_VERIFY                          (0x01)
#define UCBioAPI_FIR_PURPOSE_IDENTIFY                        (0x02)
#define UCBioAPI_FIR_PURPOSE_ENROLL                          (0x03)
#define UCBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY    (0x04)
#define UCBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY  (0x05)
#define UCBioAPI_FIR_PURPOSE_AUDIT                           (0x06)
#define UCBioAPI_FIR_PURPOSE_UPDATE                          (0x10)
```

■ UCBioAPI_FIR_QUALITY

Prototype:

```
typedef UCBioAPI_UINT16   UCBioAPI_FIR_QUALITY;
```

Description:

It represents the quality value of FIR. The range of allowed values is 0~100. The higher the value is, the better quality of fingerprint data is.

■ UCBioAPI_FIR_PRIVILEGE

Prototype:

```
typedef UCBioAPI_UINT16   UCBioAPI_FIR_PRIVILEGE;
```

Description:

It represents the data rights of FIR. Allowed values are shown below. This value is used only for reference and it does not have any effect on authentication.

```
#define UCBioAPI_FIR_PRIVILEGE_NOT_USED      (0)
#define UCBioAPI_FIR_PRIVILEGE_LOWEST        (1)
#define UCBioAPI_FIR_PRIVILEGE_LOWER         (2)
#define UCBioAPI_FIR_PRIVILEGE_LOW           (3)
#define UCBioAPI_FIR_PRIVILEGE_BELOW_NORMAL  (4)
#define UCBioAPI_FIR_PRIVILEGE_NORMAL        (5)
#define UCBioAPI_FIR_PRIVILEGE_ABOVE_NORMAL  (6)
#define UCBioAPI_FIR_PRIVILEGE_HIGH          (7)
```

```
#define UCBioAPI_FIR_PRIVILEGE_HIGHER        (8)
#define UCBioAPI_FIR_PRIVILEGE_HIGHEST       (9)
```

■ **UCBioAPI_FIR_DATE**

Prototype:

```
typedef struct ucbioapi_fir_date {
   UCBioAPI_UINT16     Year;    // 0 = 2000 / 1 = 2001 / ...
   UCBioAPI_UINT8      Month;
   UCBioAPI_UINT8      Day;
} UCBioAPI_FIR_DATE;
```

Description:

It is the structure that stores data date information of FIR. Since that value starting from 2000 is entered in Year, entering 8 is enough for year 2008. Used in the UCBioAPI_FIR_OPTIONAL_DATA structure, it can be used for authentication when using the UCBioAPI_VerifyMatchEx function.

■ **UCBioAPI_FIR_UUID_INFO**

Prototype:

```
typedef struct ucbioapi_fir_ott_info {
   UCBioAPI_UINT32     Index;
   UCBioAPI_UINT8      UUID[16];
} UCBioAPI_FIR_UUID_INFO, *UCBioAPI_FIR_UUID_INFO_PTR;
```

Description:

It is the structure that stores unique UUID (Universally Unique Identifier) information of FIR. Generally it is not used. But used in UCBioAPI_FIR_OPTIONAL_DATA structure, it is used for authentication when using the UCBioAPI_VerifyMatchEx function.

■ **UCBioAPI_OPTIONAL_DATA_TYPE**

Prototype:

```
typedef UCBioAPI_UINT32  UCBioAPI_OPTIONAL_DATA_TYPE;
```

Description:

When authenticating using the UCBioAPI_VerifyMatchEx function, it is the type that determines which of optional data is used as additional authentication data. It means each of fields of the UCBioAPI_FIR_OPTIONAL_DATA structure. Each of values can be designated repeatedly through the OR operator. Allowed values are shown below.

```
#define UCBioAPI_OPTIONAL_DATA_TYPE_UUID            (0x00000001)
#define UCBioAPI_OPTIONAL_DATA_TYPE_PIN1            (0x00000002)
#define UCBioAPI_OPTIONAL_DATA_TYPE_PIN2            (0x00000004)
#define UCBioAPI_OPTIONAL_DATA_TYPE_PRIVILEGE       (0x00000008)
#define UCBioAPI_OPTIONAL_DATA_TYPE_SITEID          (0x00000010)
#define UCBioAPI_OPTIONAL_DATA_TYPE_ISSUEDATE       (0x00000020)
#define UCBioAPI_OPTIONAL_DATA_TYPE_EXPIREDATE      (0x00000040)
#define UCBioAPI_OPTIONAL_DATA_TYPE_ALL             (0xffffffff)
```

■ **UCBioAPI_FIR_OPTIONAL_DATA**

Prototype:

```
typedef struct ucbioapi_fir_optional_data {
   UCBioAPI_UINT32              Length;
   UCBioAPI_FIR_UUID_INFO       UUIDInfo;
   UCBioAPI_UINT32              PIN1;
   UCBioAPI_UINT32              PIN2;
   UCBioAPI_UINT32              Privilege;
   UCBioAPI_UINT32              SiteID;
   UCBioAPI_FIR_DATE            IssueDate;
   UCBioAPI_FIR_DATE            ExpireDate;
   UCBioAPI_UINT32              Reserved;
} UCBioAPI_FIR_OPTIONAL_DATA, *UCBioAPI_FIR_OPTIONAL_DATA_PTR;
```

Description:

When authenticating using the UCBioAPI_VerifyMatchEx function, it is the structure for optional data used as additional data. Each of fields is described below.

Length:

As the length of the structure, it has the value of sizeof(UCBioAPI_FIR_OPTIONAL_DATA).

UUIDInfo:

Structure with UUID value of FIR. Refer to UCBioAPI_FIR_UUID_INFO description.

PIN1, PIN2:

Value that can split and store Personal Identification Number.

Privilege:

Authority value of FIR. If authority level is low during authentication, it is possible not to allow authentication.

SiteID:

ID of the site where FR data are used is designated and its use can be allowed only in specific sites.

IssueDate, ExporeDate:

Date of FIR creation and allowed dates for use can be designated.

Reserved:

Reserved area

■ **UCBioAPI_FIR_HEADER**

Prototype:

```
typedef struct ucbioapi_fir_header {
   UCBioAPI_UINT32              Length;
   UCBioAPI_UINT32              DataLength;
   UCBioAPI_FIR_VERSION         Version;
   UCBioAPI_FIR_DATA_TYPE       DataType;
   UCBioAPI_FIR_PURPOSE         Purpose;
   UCBioAPI_FIR_QUALITY         Quality;
   UCBioAPI_FIR_OPTIONAL_DATA   OptionalData;
   UCBioAPI_UINT32              Reserved;
} UCBioAPI_FIR_HEADER, *UCBioAPI_FIR_HEADER_PTR;
```

Description:

It is the structure that stores header information of FR data. It is used in the UCBioAPI_FIR structure. Each of fields is described below.

Length:

As the length of the structure, it has the value of sizeof(UCBioAPI_FIR_HEADER).

DataLength:

It has the value of the length that stores real FIR data.

Version:

It has FIR data version information. Refer to UCBioAPI_FIR_VERSION.

DataType:

It has the type of FIR data. Refer to UCBioAPI_FIR_DATA_TYPE.

Purpose:

It has the purpose of FIR data. Refer to UCBioAPI_FIR_PURPOSE.

Quality:

It has the quality value of FIR data. Refer to UCBioAPI_FIR_QUALITY.

OptionalData:

It is the structure that has the additional authentication information value of FIR data. Refer to UCBioAPI_FIR_OPTIONAL_DATA.

Reserved:

Reserved area.

■ **UCBioAPI_FIR_DATA**

Prototype:

```
typedef UCBioAPI_UINT8   UCBioAPI_FIR_DATA;
```

Description:

It represents the type of a real data block of FIR.

■ **UCBioAPI_FIR_FORMAT**

Prototype:

```
typedef UCBioAPI_UINT32  UCBioAPI_FIR_FORMAT;
```

Description:

It represents the data format of FIR. Allowed values are shown below. If the structure of FIR data is changed later, this format value is changed to keep lower level compatibility. Currently, only standard format is supported.

```
#define UCBioAPI_FIR_FORMAT_STANDARD     (1)
```

■ **UCBioAPI_FIR**

Prototype:

```
typedef struct ucbioapi_fir {
   UCBioAPI_FIR_FORMAT    Format;
   UCBioAPI_FIR_HEADER    Header;
   UCBioAPI_FIR_DATA*     Data;
} UCBioAPI_FIR, *UCBioAPI_FIR_PTR;
```

Description:

It is the structure that represents FIR data. It stores binary data of real FIR.

Format:

Data format of FIR. Refer to UCBioAPI_FIR_FORMAT.

Header:

Header that includes various informations of FIR. Refer to UCBioAPI_FIR_HEADER.

Data:

Binary block that includes real data of FIR. The length value of this data is recorded in DataLength of Header.

■ **UCBioAPI_FIR_PAYLOAD**

Prototype:

```
typedef struct ucbioapi_fir_payload {
   UCBioAPI_UINT32        Length;
   UCBioAPI_UINT8*        Data;
} UCBioAPI_FIR_PAYLOAD, *UCBioAPI_FIR_PAYLOAD_PTR;
```

Description:

It is the structure that stores Payload information. During fingerprint registration, a specific value of a user can be encrypted and inserted into the inside of FIR. That value can be brought back and read later when user authentication succeeds. UCBioAPI_FIR_PAYLOAD is the structure that stores that information. Since Payload data can be used as any data, it can be used to obtain a specific fixed value during user authentication.

Length

The length of Payload data.

Data:

Real Payload data block.

■ UCBioAPI_HANDLE / UCBioAPI_HANDLE_PTR

Prototype:

```
typedef UCBioAPI_UINT  UCBioAPI_HANDLE;
typedef UCBioAPI_UINT* UCBioAPI_HANDLE_PTR;
```

Description:

Various Handle values used in UCBioAPI are defined.

If no Handle value is available, UCBioAPI_INVALID_HANDLE(0) value is used.

```
#define UCBioAPI_INVALID_HANDLE      (0)
```

■ UCBioAPI_FIR_HANDLE / UCBioAPI_FIR_HANDLE_PTR

Prototype:

```
typedef UCBioAPI_UINT  UCBioAPI_FIR_HANDLE;
typedef UCBioAPI_UINT* UCBioAPI_FIR_HANDLE_PTR;
```

Description:

The Handle value of FIR data is defined.

■ UCBioAPI_FIR_SECURITY_LEVEL

Prototype:

```
typedef UCBioAPI_UINT32   UCBioAPI_FIR_SECURITY_LEVEL;
```

Description:

An authentication security level of FIR data is designated. Allowed values are shown below.

```
#define UCBioAPI_FIR_SECURITY_LEVEL_LOWEST          (1)
#define UCBioAPI_FIR_SECURITY_LEVEL_LOWER           (2)
#define UCBioAPI_FIR_SECURITY_LEVEL_LOW             (3)
#define UCBioAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL    (4)
#define UCBioAPI_FIR_SECURITY_LEVEL_NORMAL          (5)
#define UCBioAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL    (6)
#define UCBioAPI_FIR_SECURITY_LEVEL_HIGH            (7)
#define UCBioAPI_FIR_SECURITY_LEVEL_HIGHER          (8)
#define UCBioAPI_FIR_SECURITY_LEVEL_HIGHEST         (9)
```

■ UCBioAPI_TEMPLATE_FORMAT

Prototype:

```
typedef UCBioAPI_UINT32   UCBioAPI_TEMPLATE_FORMAT;
```

Description:

An Template format of FIR data is designated. Allowed values are shown below.

```
#define UCBioAPI_TEMPLATE_FORMAT_UNION400           (0)
#define UCBioAPI_TEMPLATE_FORMAT_ISO500             (1)
#define UCBioAPI_TEMPLATE_FORMAT_ISO600             (2)
```

■ UCBioAPI_INIT_INFO_0

Prototype:

```
typedef struct ucbioapi_init_info_0 {
   UCBioAPI_UINT32               StructureType;
   UCBioAPI_UINT32               MaxFingersForEnroll;
   UCBioAPI_UINT32               NecessaryEnrollNum;
   UCBioAPI_UINT32               SamplesPerFinger;
   UCBioAPI_UINT32               DefaultTimeout;
   UCBioAPI_FIR_SECURITY_LEVEL   SecurityLevelForEnroll;
   UCBioAPI_FIR_SECURITY_LEVEL   SecurityLevelForVerify;
   UCBioAPI_FIR_SECURITY_LEVEL   SecurityLevelForIdentify;
   UCBioAPI_TEMPLATE_FORMAT      TemplateFormat;
```

```
    UCBioAPI_UINT32              Reserved1;
    UCBioAPI_UINT32              Reserved2;
} UCBioAPI_INIT_INFO_0, *UCBioAPI_INIT_INFO_PTR_0;
```

Description:

It is the structure that stores the basic initial setting values of UCBioAPI SDK. Each of these values is described below. It is used in the UCBioAPI_GetInitInfo / UCBioAPI_SetInitInfo function.

StructureType:

Structure type. If an initial setting value is added later, an increment can be made as structure 1, 2, .. Currently, only 0 is used.

MaxFingersForEnroll:

During registration, the maximum number of fingers allowed for registration is designated. Here, if the designated number of fingers is registered, no more fingers can be registered. The default value is 10.

NecessaryEnrollNum:

The minimum number of fingers to be registered during registration is designated. If this value is set as 2, at least 2 fingers need to be registered to complete a fingerprint registration process. Therefore, this value must be less than or equal to the MaxFingersForEnroll value. The default value is 1.

SamplesPerFinger:

The number of samples per finger is designated. Currently, it is fixed at 2 and modification is not allowed.

DefaultTimeout:

During fingerprint registration and authentication, the basic Timeout value for fingerprint input waiting time is designated. The unit millisecond and set as 1,000 for 1 second. The default value is 10,000 (10 seconds).

SecurityLevelForEnroll / SecurityLevelForVerify / SecurityLevelForIdentify:

The basic security level used in fingerprint registration and authentication is designated. Refer to UCBioAPI_FIR_SECURITY_LEVEL. The basic value is used in the order of 5 / 5 / 6.

TempateFormat:

An Template format of FIR data is designated.

The default value is `UCBioAPI_TEMPLATE_FORMAT_UNION400`.

Reserved1 / Reserved2:

Reserved value.

■ **UCBioAPI_DEVICE_ID**

Prototype:

```
typedef UCBioAPI_UINT16   UCBioAPI_DEVICE_ID;
```

Description:

The ID values of devices are defined. This value is 2 bytes. The upper 1 byte means UCBioAPI_DEVICE_NAME that represents the type of each device and the lower 1 byte represents the Instance value for each device. Allowed values are shown below. To use the device used most recently from devices in the system, designate as AUTO(0x00ff).

```
#define UCBioAPI_DEVICE_ID_NONE          (0x0000)
#define UCBioAPI_DEVICE_ID_AUTO          (0x00ff)
```

■ UCBioAPI_DEVICE_NAME

Prototype:

```
typedef UCBioAPI_UINT8    UCBioAPI_DEVICE_NAME;
```

Description:

As the value used as the lower byte of UCBioAPI_DEVICE_ID, it represents the type of each device. Allowed values are shown below. If devices are added later, this value will be increased.

```
#define UCBioAPI_DEVICE_NAME_FOH01          (0x01)
#define UCBioAPI_DEVICE_NAME_FOM01          (0x02)
#define UCBioAPI_DEVICE_NAME_FOH03          (0x03)
#define UCBioAPI_DEVICE_NAME_HAM500         (0x04)
#define UCBioAPI_DEVICE_NAME_FOH01A         (0x05)
#define UCBioAPI_DEVICE_NAME_FOM01A         (0x06)
#define UCBioAPI_DEVICE_NAME_FPR02          (0x07)
#define UCBioAPI_DEVICE_NAME_FSH01RF        (0x08)
#define UCBioAPI_DEVICE_NAME_FOH01RF        (0x09)
#define UCBioAPI_DEVICE_NAME_FR100          (0x0a) // 10
#define UCBioAPI_DEVICE_NAME_FPR02LFD       (0x0b) // 11
#define UCBioAPI_DEVICE_NAME_FOH01RFL       (0x0c) // 12
#define UCBioAPI_DEVICE_NAME_FSH01SC        (0x0d) // 13
#define UCBioAPI_DEVICE_NAME_FPR02_V30      (0x0e) // 14
```

■ UCBioAPI_DEVICE_INFO_0

Prototype:

```
typedef struct ucbioapi_device_info_0 {
  UCBioAPI_UINT32     StructureType;
  UCBioAPI_UINT32     ImageWidth;
```

```
  UCBioAPI_UINT32     ImageHeight;
  UCBioAPI_UINT32     Brightness;
  UCBioAPI_UINT32     Contrast;
  UCBioAPI_UINT32     Gain;
} UCBioAPI_DEVICE_INFO_0, *UCBioAPI_DEVICE_INFO_PTR_0;
```

Description:

The structure that stores the value on device information.

It is used in the UCBioAPI_GetDeviceInfo / UCBioAPI_SetDeviceInfo function. Each of these values is described here.

StructureType:

Structure type. If an initial value is added later, an increment can be made as structure 1, 2, .. Only 0 is currently used.

ImageWidth / ImageHeight:

The size of image that can be obtained from a device. This value can be different from device to device. This value can not be changed to read-only.

Brightness / Contrast / Gain:

The brightness / contrast / gain value of device can be obtained or designated. Currently, only the value for gain is validly used, and values allowed for use are different according to devices.

■ **UCBioAPI_DEVICE_INFO_EX**

Prototype:

```
typedef struct ucbioapi_deviceinfoex {
  UCBioAPI_DEVICE_ID     NameID;
  UCBioAPI_UINT16        Instance;

  UCBioAPI_CHAR          Name[64];
  UCBioAPI_CHAR          Description[256];
  UCBioAPI_CHAR          Dll[64];
  UCBioAPI_CHAR          Sys[64];

  UCBioAPI_UINT32        Brightness;
  UCBioAPI_UINT32        Contrast;
  UCBioAPI_UINT32        Gain;

  UCBioAPI_UINT32        Reserved[8];
} UCBioAPI_DEVICE_INFO_EX, *UCBioAPI_DEVICE_INFO_EX_PTR;
```

Description:

The structure that stores more detailed value on device information.

When obtaining the device list, the value contained in this structure is passed back in the UCBioAPI_EnumerateDevice function. Each of values is described here.

NameID:

The type of device is designated. Refer to UCBioAPI_DEVICE_ID.

Instance:

Device-by-device Instance value is designated. Refer to UCBioAPI_DEVICE_ID value.

Name:

The device names are stored in string type.

Description:

Descriptions on devices are stored in string type.

Dll / Sys:

Information on DLL, Sys files used by devices are stored.

Brightness / Contrast / Gain:

The brightness / contrast / gain value of a device can be obtained or designated. Currently, only the value for gain is validly used, and values allowed for use are different according to devices.

Reserved:

Reserved value.

■ UCBioAPI_RETURN

Prototype:

```
typedef UCBioAPI_UINT32  UCBioAPI_RETURN;
```

Description:

Values returned by functions in UCBioAPI SDK are defined. In general, error values of UCBioAPI SDK are included. For detailed error values, refer to the error definition.

■ UCBioAPI_FIR_TEXTENCODE

Prototype:

```
typedef struct ucbioapi_fir_textencode {
   UCBioAPI_BOOL          IsWideChar;
   UCBioAPI_CHAR_PTR      TextFIR;
} UCBioAPI_FIR_TEXTENCODE, *UCBioAPI_FIR_TEXTENCODE_PTR;
```

Description:

The structure that stores values representing FIR data in text type. FIR data can be represented in either binary type or text type and two types produce the same result. If it

is difficult to use binary type data, it is used to conveniently store FIR in text string type. It can be obtained using a function such as UCBioAPI_GetTextFIRFromHandle.

IsWideChar:
It represents if TextFIR consists of Unicode 2 byte character string.
TextFIR:
Real FIR data are inserted in text string type.

■ **UCBioAPI_INPUT_FIR_FORM**

Prototype:

```
typedef UCBioAPI_UINT8   UCBioAPI_INPUT_FIR_FORM;
```

Description:

The type of structure used when passing FIR data as a function argument is defined. FIR data can be used as three different types in total such as FIR Handle, Binary Text and Text FIR and its type can be designated. Allowed values for use are shown below.

```
#define UCBioAPI_FIR_FORM_HANDLE       (0x02)
#define UCBioAPI_FIR_FORM_FULLFIR      (0x03)
#define UCBioAPI_FIR_FORM_TEXTENCODE   (0x04)
```

■ **UCBioAPI_INPUT_FIR**

Prototype:

```
typedef struct ucbioapi_input_fir {
  UCBioAPI_INPUT_FIR_FORM  Form;
  union {
    UCBioAPI_FIR_HANDLE_PTR          FIRinBSP;
    UCBioAPI_VOID_PTR                FIR;
    UCBioAPI_FIR_TEXTENCODE_PTR      TextFIR;
  } InputFIR;
} UCBioAPI_INPUT_FIR, *UCBioAPI_INPUT_FIR_PTR;
```

Description:

The structure used when passing FIR data as a function argument. FIR data can be used as three different types in total such as FIR Handle, Binary FIR and Text FIR. This structure is used to represent them simultaneously as one input value.

Form:
The type of FIR data that this structure has is designated. Refer to

UCBioAPI_INPUT_FIR_FORM.

InputFIR:

The union structure that designate real FIR data. FIR Handle, Binary FIR and Text FIR values are stored as one identical address pointer for use.

■ UCBioAPI_WINDOW_CALLBACK_PARAM_0

Prototype:

```
typedef struct ucbioapi_window_callback_param_0 {
   UCBioAPI_UINT32        dwQuality;
   UCBioAPI_UINT8*        lpImageBuf;
   UCBioAPI_UINT32        dwDeviceError;

   UCBioAPI_UINT32        dwReserved[8];
   UCBioAPI_VOID_PTR      lpReserved;
} UCBioAPI_WINDOW_CALLBACK_PARAM_0,
*UCBioAPI_WINDOW_CALLBACK_PARAM_PTR_0;
```

Description:

The structure passed as the first argument of the Callback function called during fingerprint acquisition and authentication.

dwQuality:

It has the fingerprint quality value of the current image.

lpImageBuf:

It has the buffer pointer of the fingerprint image currently acquired.

dwDeviceError:

If there is an error value occurred from the current device, it has that value.

dwReserved / lpReserved:

Reserved value.

■ UCBioAPI_WINDOW_CALLBACK_PARAM_1

Prototype:

```
typedef struct ucbioapi_window_callback_param_1 {
   UCBioAPI_UINT32        dwResult;

   UCBioAPI_UINT32        dwStartTime;
   UCBioAPI_UINT32        dwCapTime;
   UCBioAPI_UINT32        dwEndTime;
```

```
    UCBioAPI_UINT32        Reserved[8];
    UCBioAPI_VOID_PTR      lpReserved;
} UCBioAPI_WINDOW_CALLBACK_PARAM_1,
*UCBioAPI_WINDOW_CALLBACK_PARAM_PTR_1;
```

Description:

The structure passed as the first argument of the Callback function called during fingerprint registration and function termination.

dwResult:

During function termination, it stores the value to be returned by that function. Also, it can have the value of an event occurred according to a registration process during fingerprint registration. Each of values can store values defined in the error definition.

dwStartTime / dwCapTime / dwEndTime:

During function termination, various time values can be found out. The time that a function was called / The starting time of real fingerprint acquisition / The time that acquisition was terminated are stored sequentially in order.

Reserved / lpReserved:

Reserved value.

■ **UCBioAPI_WINDOW_CALLBACK_0 / UCBioAPI_WINDOW_CALLBACK_1**

Prototype:

```
typedef UCBioAPI_RETURN (WINAPI* UCBioAPI_WINDOW_CALLBACK_0)
(UCBioAPI_WINDOW_CALLBACK_PARAM_PTR_0, UCBioAPI_VOID_PTR);

typedef UCBioAPI_RETURN (WINAPI* UCBioAPI_WINDOW_CALLBACK_1)
(UCBioAPI_WINDOW_CALLBACK_PARAM_PTR_1, UCBioAPI_VOID_PTR);
```

Description:

Definition of the function pointer to the Callback function called during fingerprint acquisition and registration.

Each of functions can be designated at CaptureCallBackInfo and FinishCallBackInfo of UCBioAPI_WINDOW_OPTION. Information appropriate for each Callback function is passed as the first argument and the value of the pointer that a user passes is passed as the second argument.

The Callback function returns 0. If a value other than 0 is returned, the function currently in use is terminated.

■ **UCBioAPI_CALLBACK_INFO_0 / UCBioAPI_CALLBACK_INFO_1**

Prototype:

```
typedef struct ucbioapi_callback_info_0 {
   UCBioAPI_UINT32              CallBackType;
   UCBioAPI_WINDOW_CALLBACK_0   CallBackFunction;
   UCBioAPI_VOID_PTR            UserCallBackParam;
} UCBioAPI_CALLBACK_INFO_0, *UCBioAPI_CALLBACK_INFO_PTR_0;

typedef struct ucbioapi_callback_info_1 {
   UCBioAPI_UINT32              CallBackType;
   UCBioAPI_WINDOW_CALLBACK_1   CallBackFunction;
   UCBioAPI_VOID_PTR            UserCallBackParam;
} UCBioAPI_CALLBACK_INFO_1, *UCBioAPI_CALLBACK_INFO_PTR_1;
```

Description:

The structure to designate information on the Callback function to be called during fingerprint acquisition.

CallBackType:

The type of the Callback function is designated. This value is always 0 in UCBioAPI_CALLBACK_INFO_0 and 1 in UCBioAPI_CALLBACK_INFO_1.

CallBackFunction:

The function to be called as real Callback is designated.

UserCallBackParam:

The user argument to be passed as the second argument of the Callback function is designated.

■ UCBioAPI_WINDOW_STYLE

Prototype:

```
typedef UCBioAPI_UINT32  UCBioAPI_WINDOW_STYLE;
```

Description:

The Window style of the user interface used in fingerprint registration and authentication is designated. This value is used in the UCBioAPI_WINDOW_OPTION structure. Allowed values for use are shown below.

```
#define UCBioAPI_WINDOW_STYLE_POPUP          (0)
#define UCBioAPI_WINDOW_STYLE_INVISIBLE      (1)

#define UCBioAPI_WINDOW_STYLE_NO_FPIMG       (0x00010000)
#define UCBioAPI_WINDOW_STYLE_NO_WELCOME     (0x00020000)
```

```
#define UCBioAPI_WINDOW_STYLE_NO_TOPMOST      (0x00040000)
```

UCBioAPI_WINDOW_STYLE_POPUP:
UCBioAPI_WINDOW_STYLE_INVISIBLE:
If pop-up is designated, a new window is generally opened to perform fingerprint registration and authentication. However, an invisible value can be designated for the UCBioAPI_Capture and UCBioAPI_Verify function and then fingerprint input or authentication can be performed while not displaying UI on the screen. Also, assuming that the value of FingerWnd from values of the UCBioAPI_WINDOW_OPTION structure is not NULL, fingerprint image can be displayed on that Window when it is designated as invisible.

3 styles in below are values that can be designated repeatedly using the OR operator. Each of values has the following meaning.

UCBioAPI_WINDOW_STYLE_NO_FPIMG:
This style is designated not to display fingerprint image on the screen for security reason.
UCBioAPI_WINDOW_STYLE_NO_WELCOME:
.The first page 'Welcome Page' is not displayed in the fingerprint registration UI.
UCBioAPI_WINDOW_STYLE_NO_TOPMOST:
Currently, all UIs are opened as the most upper Window. This style is designated to open them as a standard Window.

■ **UCBioAPI_WINDOW_OPTION**

Prototype:

```
typedef struct ucbioapi_window_option {
   UCBioAPI_UINT32               Length;
   UCBioAPI_WINDOW_STYLE         WindowStyle;
   UCBioAPI_HWND                 ParentWnd;
   UCBioAPI_HWND                 FingerWnd;
   UCBioAPI_CALLBACK_INFO_0      CaptureCallBackInfo;
   UCBioAPI_CALLBACK_INFO_1      FinishCallBackInfo;
   UCBioAPI_CHAR_PTR             CaptionMsg;
   UCBioAPI_CHAR_PTR             CancelMsg;
   UCBioAPI_WINDOW_OPTION_PTR_2  Option2;
} UCBioAPI_WINDOW_OPTION, *UCBioAPI_WINDOW_OPTION_PTR;
```

Description:
The structure that designates various informations of a user interface used in fingerprint registration and authentication. Each of values is described below.

Length:
The length of the structure. It has the value of sizeof(UCBioAPI_WINDOW_OPTION).
WindowStyle:
The value of UI Window style is designated. Refer to UCBioAPI_WINDOW_STYLE.
The default value is UCBioAPI_WINDOW_STYLE_POPUP(0).
ParentWnd:
When the fingerprint registration and authentication Window opens, the Handle of the parent Window that is the base of that Window is designated. The default value is NULL.
FingerWnd:
If the value of WindowStyle is UCBioAPI_WINDOW_STYLE_INVISIBLE(1), the Handle of a specific Window that draws fingerprint image is designated. The default value is NULL.
CaptureCallBackInfo:
The Callback function to be called at every fingerprint acquisition is designated. Refer to UCBioAPI_CALLBACK_INFO_0.
FinishCallBackInfo:
The Callback function to be called at every fingerprint registration and function termination is designated. Refer to UCBioAPI_CALLBACK_INFO_1.
CaptionMsg:
When a user selects Cancel during fingerprint registration, a message to be displayed in Caption of Window to show the cancel message is designated. The default value is NULL.
CancelMsg:
When a user selects Cancel during fingerprint registration, a cancel message to be displayed is designated. The default value is NULL.
Option2:
Additional Window option values are designated. Refer to UCBioAPI_WINDOW_OPTION_2 for detailed values. The default value is NULL.

■ UCBioAPI_WINDOW_OPTION_2

Prototype:

```
typedef struct ucbioapi_window_option_2 {
   UCBioAPI_UINT8         FPForeColor[3];
   UCBioAPI_UINT8         FPBackColor[3];
   UCBioAPI_UINT8         DisableFingerForEnroll[10];
   UCBioAPI_UINT32        Reserved1[4];
   UCBioAPI_VOID_PTR      Reserved2;
} UCBioAPI_WINDOW_OPTION_2, *UCBioAPI_WINDOW_OPTION_PTR_2;
```

Description:

The structure that designates additional Window option values. Each of values is described below.

FPForeColor:
The fingerprint color value of fingerprint image to be displayed on the screen is designated as RGB value.
FPBackColor:
The fingerprint background value of fingerprint image to be displayed on the screen is designated as RGB value.
DisableFingerForEnroll
During fingerprint registration, fingers prohibited for registration can be designated. As the array value that can store 10 values in total, registration status of each finger value is designated as 0 or 1.
It increases from index number 1 in the order of the thumb, index, middle, ring and little of the right hand and increases from index number 6 in the order of the thumb, index, middle, ring and little of the left hand. Index number 0 is not used.
For example, to prohibit the left thumb from registration, set as shown here.

```
DisableFingerForEnroll[UCBioAPI_FINGER_ID_LEFT_THUMB] = 1;
```

But, in the fingerprint correction mode, registration status for a finger prohibited for use is displayed if it is an already registered finger.

Reserved1 / Reserved2
Reserved value.

■ UCBioAPI_TEMPLATE_TYPE

Prototype:

```
typedef UCBioAPI_UINT32     UCBioAPI_TEMPLATE_TYPE;
```

Description:
When converting FR data into template data, allowed template type values are designated. Allowed values are shown below. If an additional type becomes available, this value can be added.

```
#define UCBioAPI_TEMPLATE_TYPE_SIZE400        (400)
#define UCBioAPI_TEMPLATE_TYPE_SIZE500        (500)
#define UCBioAPI_TEMPLATE_TYPE_SIZE600        (600)
#define UCBioAPI_TEMPLATE_TYPE_SIZE800        (800)
#define UCBioAPI_TEMPLATE_TYPE_SIZE320        (320)
```

```
#define UCBioAPI_TEMPLATE_TYPE_SIZE256        (256)
#define UCBioAPI_TEMPLATE_TYPE_FMR            (1)
#define UCBioAPI_TEMPLATE_TYPE_ANSI                   (2)
```

SIZE400 / SIZE800 / SIZE320 / SIZE256 mean templates with the given size. If fingerprints are stored in a limited storage space such as the smart card, data such as SIZE256 can be used. (But, as size gets smaller, authentication rate is expected to decrease.)
FMR type is the standard data format for fingerprint data.

■ UCBioAPI_FINGER_ID

Prototype:

```
typedef UCBioAPI_UINT8      UCBioAPI_FINGER_ID;
```

Description:

ID values of fingers are designated. Allowed values are shown below.

```
#define UCBioAPI_FINGER_ID_UNKNOWN          (0)
#define UCBioAPI_FINGER_ID_RIGHT_THUMB      (1)
#define UCBioAPI_FINGER_ID_RIGHT_INDEX      (2)
#define UCBioAPI_FINGER_ID_RIGHT_MIDDLE     (3)
#define UCBioAPI_FINGER_ID_RIGHT_RING       (4)
#define UCBioAPI_FINGER_ID_RIGHT_LITTLE     (5)
#define UCBioAPI_FINGER_ID_LEFT_THUMB       (6)
#define UCBioAPI_FINGER_ID_LEFT_INDEX       (7)
#define UCBioAPI_FINGER_ID_LEFT_MIDDLE      (8)
#define UCBioAPI_FINGER_ID_LEFT_RING        (9)
#define UCBioAPI_FINGER_ID_LEFT_LITTLE      (10)
#define UCBioAPI_FINGER_ID_MAX              (11)
```

UCBioAPI_FINGER_ID_UNKNOWN is used when specific finger information is not known during capture.

■ UCBioAPI_MATCH_OPTION_0

Prototype:

```
typedef struct ucbioapi_match_option_0 {
  UCBioAPI_UINT8        StructureType;
  UCBioAPI_UINT8        NoMatchFinger[UCBioAPI_FINGER_ID_MAX];
  UCBioAPI_UINT32       Reserved[8];
} UCBioAPI_MATCH_OPTION_0, *UCBioAPI_MATCH_OPTION_PTR_0;
```

Description:
The structure to store information used for detailed authentication in the UCBioAPI_VerifyMatchEx function.

StructureType:
Structure type. If initial setting values are added later, an increment can be made as structure 1, 2, .. Only 0 is currently used.

NoMatchFinger:
ID of a finger not used in authentication can be designated. Even though all 10 fingers are registered in FIR, this value is designated to exclude some fingers for authentication. Also, some samples for each finger can be excluded for authentication.
The first sample is excluded if 1 is designated and the second sample is excluded if 2 is designated. If 3 is designated, that finger is not used in authentication.
To exclude the first sample of the right thumb and the second sample of the left thumb, follow the next procedure.

```
NoMatchFinger[UCBioAPI_FINGER_ID_RIGHT_THUMB] = 1;
NoMatchFinger[UCBioAPI_FINGER_ID_LEFT_THUMB] = 2;
```

Reserved:
Reserved value.

5.1.3. Export/Import functions related types

Declaration is made in UCBioAPI_ExportType.h and types related to data conversion such as export and import are defined.

■ **UCBioAPI_TEMPLATE_BLOCK**

Prototype:

```
typedef struct ucbioapi_template_block {
   UCBioAPI_UINT32        Length;
   UCBioAPI_UINT8*        Data;
} UCBioAPI_TEMPLATE_BLOCK, *UCBioAPI_TEMPLATE_BLOCK_PTR;
```

Description:

The structure to store 1 template information.

Length:

The length of data. Note that it is not the length of the structure unlike other structures.

Data:

Binary data block that stores template data. Stored data vary according to the TemplateType value of the UCBioAPI_EXPORT_DATA structure.

■ **UCBioAPI_FINGER_BLOCK**

Prototype:

```
typedef struct ucbioapi_finger_block {
   UCBioAPI_UINT32              Length;
   UCBioAPI_FINGER_ID           FingerID;
   UCBioAPI_TEMPLATE_BLOCK_PTR  TemplateInfo;
} UCBioAPI_FINGER_BLOCK, *UCBioAPI_FINGER_BLOCK_PTR;
```

Description:

The structure to store 1 finger information. Several template informations can be stored in one finger.

Length:

The length of the structure. It has the value of sizeof(UCBioAPI_FINGER_BLOCK).

FingerID:

It has the finger ID value. Refer to UCBioAPI_FINGER_ID.

TemplateInfo:

The structure array that stores template information.

To store a large number of template informations, UCBioAPI_TEMPLATE_BLOCK can exist in

several arrays. The number of arrays is stored in the SamplesPerFinger value in the UCBioAPI_EXPORT_DATA structure.

■ **UCBioAPI_EXPORT_DATA**

Prototype:

```
typedef struct ucbioapi_export_data {
   UCBioAPI_UINT32              Length;
   UCBioAPI_TEMPLATE_TYPE       TemplateType;
   UCBioAPI_UINT8               FingerNum;
   UCBioAPI_FINGER_ID           DefaultFingerID;
   UCBioAPI_UINT8               SamplesPerFinger;
   UCBioAPI_UINT8               Reserved;
   UCBioAPI_FINGER_BLOCK_PTR    FingerInfo;
} UCBioAPI_EXPORT_DATA, *UCBioAPI_EXPORT_DATA_PTR;
```

Description:

The structure to store template information for one FIR. Several finger informations can be stored in one FIR.

Length:

The length of the structure. It has the value of sizeof(UCBioAPI_EXPORT_DATA).

TemplateType:

Data type of stored template. Refer to UCBioAPI_TEMPLATE_TYPE.

FingerNum:

The total number of fingers is designated. FingerInfo informations as many as the number designated here are stored in array.

DefaultFingerID:

The representative finger ID is designated.

SamplesPerFinger:

The number of templates per finger is designated. TemplateInfo informations in FingerInfo information as many as the number designated here are stored in array.

Reserved:

Reserved area.

FingerInfo:

The structure array that stores finger information.

To store a large number of finger informations, UCBioAPI_FINGER_BLOCK can exist in several arrays. The number of arrays is stored in the FingerNum value.

■ **UCBioAPI_IMAGE_DATA**

Prototype:

```
typedef struct ucbioapi_image_data {
   UCBioAPI_UINT32        Length;
   UCBioAPI_UINT8*        Data;
} UCBioAPI_IMAGE_DATA, *UCBioAPI_IMAGE_DATA_PTR;
```

Description:

The structure to store one fingerprint image.

Length:

The length of the structure. It has the value of sizeof(UCBioAPI_IMAGE_DATA).

Data:

Binary data block that stores fingerprint image data. Image is stored as the image format in raw type. The length of this data block is the same as the multiplication value between the ImageWidth and ImageHeight value of the UCBioAPI_EXPORT_AUDIT_DATA structure. That is, Length of Data = ImageWidth * ImageHeight.

■ UCBioAPI_AUDIT_DATA

Prototype:

```
typedef struct ucbioapi_audit_data {
   UCBioAPI_UINT32              Length;
   UCBioAPI_UINT8               FingerID;
   UCBioAPI_IMAGE_DATA_PTR      Image;
} UCBioAPI_AUDIT_DATA, *UCBioAPI_AUDIT_DATA_PTR;
```

Description:

The structure to store fingerprint image for one finger. Several fingerprint image informations can be stored in one finger.

Length:

The length of the structure. It has the value of sizeof(UCBioAPI_AUDIT_DATA).

FingerID:

It has the finger ID value. Refer to UCBioAPI_FINGER_ID.

Image:

The structure array that stores fingerprint image information.

To store a large number of fingerprint image informations, UCBioAPI_IMAGE_DATA can exist in several arrays. The number of arrays is stored in the SamplesPerFinger in the UCBioAPI_EXPORT_AUDIT_DATA structure.

■ **UCBioAPI_EXPORT_AUDIT_DATA**

Prototype:

```
typedef struct ucbioapi_export_audit_data {
   UCBioAPI_UINT32              Length;
   UCBioAPI_UINT8               FingerNum;
   UCBioAPI_UINT8               SamplesPerFinger;
   UCBioAPI_UINT32              ImageWidth;
   UCBioAPI_UINT32              ImageHeight;
   UCBioAPI_AUDIT_DATA_PTR      AuditData;
   UCBioAPI_UINT32              Reserved;
} UCBioAPI_EXPORT_AUDIT_DATA, *UCBioAPI_EXPORT_AUDIT_DATA_PTR;
```

Description:

The structure to store fingerprint image information for one Audit FIR. Several finger image informations can be stored in one Audit FIR.

Length:

The length of the structure. It has the value of sizeof(UCBioAPI_EXPORT_AUDIT_DATA).

FingerNum:

The total number of fingers is designated. AuditData informations as many as the number designated are stored in array.

SamplesPerFinger:

The number of fingerprint images per finger is stored. Image informations in AuditData information as many as the number designated here are stored in array.

ImageWidth / ImageHeight:

The size of image is designated.

AuditData:

The structure array that stores finger image information.

To store a large number of finger image informations, UCBioAPI_AUDIT_DATA can exist in several arrays. The number of arrays is stored in the FingerNum value.

Reserved:

Reserved value.

5.1.4. FastSearch functions related types

Declaration is made in UCBioAPI_FastSearchType.h and types related to FastSearch are defined.

■ **UCBioAPI_FASTSEARCH_INIT_INFO_0**

Prototype:

```
typedef struct ucbioapi_fastsearch_init_info_0 {
   UCBioAPI_UINT32              StructureType;
   UCBioAPI_UINT32              UseGroupMatch;
   UCBioAPI_UINT32              MatchMethod;
   UCBioAPI_UINT32              Researved1;
   UCBioAPI_UINT32              Researved2;
   UCBioAPI_UINT32              Researved3;
   UCBioAPI_UINT32              Researved4;
   UCBioAPI_UINT32              Researved5;
   UCBioAPI_UINT32*             Researved6;
} UCBioAPI_FASTSEARCH_INIT_INFO_0,
*UCBioAPI_FASTSEARCH_INIT_INFO_PTR_0;
```

Description:

The structure that stores the basic initial setting value of FastSearch.
It is used in the UCBioAPI_GetFastSearchInitInfo / UCBioAPI_SetFastSearchInitInfo function. Each of values is described below.

StructureType:

Structure type. If initial setting values are added later, an increment can be made as structure 1, 2, .. Only 0 is currently used.

UseGroupMatch:

When performing authentication, decision is made if group unit authentication is performed. Authentication is performed sequentially following the order in DB if this value is 0 and group unit authentication is performed if this value is 1. The default value is 1.

MatchMethod:

The method to perform authentication is determined. If this value is 0, an authentication level is set and authentication ends immediately when it goes over that level. If this value is 1, the highest point authentication method is used and it is a method to find the value with the highest authentication level. The default value is 0. It is recommended to set this value as 0 for authentication.

Reserved1 ~ Reserved6:

Reserved value.

■ **UCBioAPI_FASTSEARCH_INIT_INFO_0**

Prototype:

```
typedef struct ucbioapi_fastsearch_fp_info {
   UCBioAPI_UINT32        ID;
   UCBioAPI_UINT8         FingerID;
   UCBioAPI_UINT8         SampleNumber;
   UCBioAPI_UINT32        Reserved1;
   UCBioAPI_UINT32        Reserved2;
} UCBioAPI_FASTSEARCH_FP_INFO, *UCBioAPI_FASTSEARCH_FP_INFO_PTR;
```

Description:

The structure that stores authentication information for each template of FastSearch. When authenticating with FastSearch, a value is stored in this structure and passed back. Each of values is described below.

ID:

User ID value.

FingerID:

Finger ID value.

SampleNumber:

Finger template number.

Reserved1 / Reserved2:

Reserved value.

■ **UCBioAPI_FASTSEARCH_SAMPLE_INFO**

Prototype:

```
typedef struct ucbioapi_fastsearch_sample_info {
   UCBioAPI_UINT32        ID;
   UCBioAPI_UINT8         SampleCount[11];
} UCBioAPI_FASTSEARCH_SAMPLE_INFO,
*UCBioAPI_FASTSEARCH_SAMPLE_INFO_PTR;
```

Description:

When FIR is added to the FastSearch DB, information of each template stored in the added FIR can be obtained. It is the structure that stores that information. It is used in the UCBioAPI_AddFIRToFastSearchDB function.

ID:

User ID value.

SamplesCount:

The number of templates for each finger is stored. It increases sequentially from index number 1 in the order of thumb, index, middle, ring and little of the right hand and from index number 6 in the order of thumb, index, middle, ring and little of the left hand. Index number 0 is not used.
For example, to obtain the number of templates stored in the right hand thumb, follow the next line.

```
int cnt = SampleCount[UCBioAPI_FINGER_ID_RIGHT_THUMB];
```

■ **UCBioAPI_FASTSEARCH_CALLBACK_PARAM_0**

Prototype:

```
typedef struct ucbioapi_fastsearch_callback_param_0 {
   UCBioAPI_UINT32              TotalCount;
   UCBioAPI_UINT32              MatchedIndex;
   UCBioAPI_UINT32              MatchedScore;
   UCBioAPI_UINT32              Reserved1;
   UCBioAPI_UINT32              Reserved2;
   UCBioAPI_UINT32              Reserved3;
   UCBioAPI_VOID_PTR            Reserved4;
} UCBioAPI_FASTSEARCH_CALLBACK_PARAM_0,
*UCBioAPI_FASTSEARCH_CALLBACK_PARAM_PTR_0;
```

Description:

It is the structure passed back as the first argument of the Callback function called at every authentication during 1:N authentication through FastSearch. Various informations are stored.

TotalCount:

The total number of fingerprints in the current DB.

MatchedIndex:

The index number of the fingerprint currently in authentication.

MatchedScore:

The authentication score of the fingerprint currently in authentication.

Reserved1 ~ Reserved4:

Reserved value.

■ **UCBioAPI_FASTSEARCH_CALLBACK_0**

Prototype:

```
typedef UCBioAPI_RETURN (WINAPI* UCBioAPI_FASTSEARCH_CALLBACK_0)
(UCBioAPI_FASTSEARCH_CALLBACK_PARAM_PTR_0, UCBioAPI_VOID_PTR);
```

Description:

When performing 1:N authentication through FastSearch, the function pointer to the Callback function called at every authentication is defined.

This function can be designated as arguments of the UCBioAPI_IdentifyFIRFromFastSearchDB function. Each UCBioAPI_FASTSEARCH_CALLBACK_PARAM_0 structure information is passed back as the first argument and the pointer value passed by a user is passed back as the second argument.

The Callback function returns 0. If a value other than 0 is returned, 1:N authentication of FastSearch in progress is terminated.

■ UCBioAPI_FASTSEARCH_CALLBACK_INFO_0

Prototype:

```
typedef struct ucbioapi_fastsearch_callback_info_0 {
   UCBioAPI_UINT32                    CallBackType;
   UCBioAPI_FASTSEARCH_CALLBACK_0     CallBackFunction;
   UCBioAPI_VOID_PTR                  UserCallBackParam;
} UCBioAPI_FASTSEARCH_CALLBACK_INFO_0,
*UCBioAPI_FASTSEARCH_CALLBACK_INFO_PTR_0;
```

Description:

The structure to store the Callback function of FastSearch.

CallBackType:

The type of the Callback function is designated. Currently, this value is always 0.

CallBackFunction:

The function actually called with Callback is designated.

UserCallBackParam:

The user argument to be passed as the second argument of the Callback function is designated.

5.1.5. SmartCard functions related types

Declaration is made in UCBioAPI_SmartCardType.h and types related to the smart card are defined.

■ **UCBioAPI_SC_USE_KEY_A / UCBioAPI_SC_USE_KEY_B**

Prototype:

```
#define UCBioAPI_SC_USE_KEY_A                (0x60)
#define UCBioAPI_SC_USE_KEY_B                (0x61)
```

Description:

The value used to designate which key between key A or key B from Mifare card keys is used.

■ **UCBioAPI_SC_LED_TOGGLE / UCBioAPI_SC_LED_NOT_TOGGLE**

Prototype:

```
#define UCBioAPI_SC_LED_TOGGLE          (1)
#define UCBioAPI_SC_LED_NOT_TOGGLE      (0)
```

Description:

When using a function related to the smart card, it designates if the function success/failure status of a device is displayed on LED or not. If UCBioAPI_SC_LED_TOGGLE is set, a success of a smart card related function changes LED of a device into blue. In other cases, it changes to red.

5.2. Error Definitions

Definition on various error values used in UCBioBSP SDK and those error values are described.
All error values are defined in the UCBioAPI_Error.h file.

5.2.1. Success

Definitions on error values used upon success.

■ **UCBioAPIERROR_NONE**

Prototype:

```
#define UCBioAPIERROR_NONE                    (0)
```

Description:

The error value available upon success. This case represents a success of the function rather than error.

5.2.2. General error definitions

They are definitions on general error values.
These error values start from the UCBioAPIERROR_BASE_GENERAL(0) value.

■ **UCBioAPIERROR_INVALID_HANDLE**

Prototype:

```
#define UCBioAPIERROR_INVALID_HANDLE          (0x0001)
```

Description:

Wrong Handle value is used.

■ **UCBioAPIERROR_INVALID_POINTER**

Prototype:

```
#define UCBioAPIERROR_INVALID_POINTER         (0x0002)
```

Description:

Wrong pointer value is used.

■ **UCBioAPIERROR_INVALID_TYPE**

Prototype:

```
#define UCBioAPIERROR_INVALID_TYPE            (0x0003)
```

Description:

Wrong type value is used. It occurs when using StructureType not supported by a function such as UCBioAPI_SetInitInfo as a function argument.

■ **UCBioAPIERROR_FUNCTION_FAIL**

Prototype:

```
#define UCBioAPIERROR_FUNCTION_FAIL           (0x0004)
```

Description:

It occurs when a function execution fails due to an occurrence of a function internal error.

■ **UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED**

Prototype:

```
#define UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED (0x0005)
```

Description:

A structure that does not match with StructureType is used. It occurs when a structure that does not correspond to StructureType requested from a function such as UCBioAPI_SetInitInfo is used as an argument.

■ **UCBioAPIERROR_ALREADY_PROCESSED**

Prototype:

```
#define UCBioAPIERROR_ALREADY_PROCESSED       (0x0006)
```

Description:

It occurs when FIR data that were entered in the UCBioAPI_Process function are already processed data.

■ **UCBioAPIERROR_EXTRACTION_OPEN_FAIL**

Prototype:

```
#define UCBioAPIERROR_EXTRACTION_OPEN_FAIL    (0x0007)
```

Description:

A case which an error occurs when initializing Engine for extraction.

■ **UCBioAPIERROR_VERIFICATION_OPEN_FAIL**

Prototype:

```
#define UCBioAPIERROR_VERIFICATION_OPEN_FAIL (0x0008)
```

Description:

A case which an error occurs when initializing Engine for authentication.

■ **UCBioAPIERROR_DATA_PROCESS_FAIL**

Prototype:

```
#define UCBioAPIERROR_DATA_PROCESS_FAIL       (0x0009)
```

Description:

An error occurs while extracting special features from fingerprint image data.

■ **UCBioAPIERROR_MUST_BE_PROCESSED_DATA**

Prototype:

```
#define UCBioAPIERROR_MUST_BE_PROCESSED_DATA (0x000a)
```

Description:

It occurs when data that have not been processed are entered into a function requiring special feature data input.

■ **UCBioAPIERROR_INTERNAL_CHECKSUM_FAIL**

Prototype:

```
#define UCBioAPIERROR_INTERNAL_CHECKSUM_FAIL (0x000b)
```

Description:

Internal validity error of FIR data. It occurs generally when data were changed arbitrarily or damaged FIR data were used.

■ **UCBioAPIERROR_ENCRYPTED_DATA_ERROR**

Prototype:

```
#define UCBioAPIERROR_ENCRYPTED_DATA_ERROR    (0x000c)
```

Description:

Encryption error of FIR data. It occurs generally when data were arbitrarily changed or encrypted data can not be recovered due to the use of damaged FIR data.

■ **UCBioAPIERROR_UNKNOWN_FORMAT**

Prototype:

```
#define UCBioAPIERROR_UNKNOWN_FORMAT          (0x000d)
```

Description:

Unknown FIR format.

■ **UCBioAPIERROR_UNKNOWN_VERSION**

Prototype:

```
#define UCBioAPIERROR_UNKNOWN_VERSION         (0x000e)
```

Description:

Unknown FIR version.

■ **UCBioAPIERROR_VALIDITY_FAIL**

Prototype:

```
#define UCBioAPIERROR_VALIDITY_FAIL          (0x000f)
```

Description:

Validity error of the UCBioBSP.dll module. It occurs generally when DLL is arbitrarily changed or unsigned DLL is used.

In UCBioBSP SDK, DLL itself examines its validity to check if modification occurs. If DLL is modified even as small as 1 byte from outside, this error occurs and SDK does not work.

■ **UCBioAPIERROR_INVALID_TEMPLATESIZE**

Prototype:

```
#define UCBioAPIERROR_INVALID_TEMPLATESIZE   (0x0010)
```

Description:

It occurs when the size of entered template is wrong during the use of the mutual conversion function between FIR data and template data.

■ **UCBioAPIERROR_INVALID_TEMPLATE**

Prototype:

```
#define UCBioAPIERROR_INVALID_TEMPLATE       (0x0011)
```

Description:

A case which wrong template data were used.

■ **UCBioAPIERROR_EXPIRED_VERSION**

Prototype:

```
#define UCBioAPIERROR_EXPIRED_VERSION        (0x0012)
```

Description:

It occurs when the valid period of the evaluation version is over during the use of the evaluation version of UCBioBSP SDK.

■ **UCBioAPIERROR_INVALID_SAMPLESPERFINGER**

Prototype:

```
#define UCBioAPIERROR_INVALID_SAMPLESPERFINGER      (0x0013)
```

Description:
It occurs when the number of templates per finger is incorrectly designated.

■ **UCBioAPIERROR_UNKNOWN_INPUTFORMAT**

Prototype:

```
#define UCBioAPIERROR_UNKNOWN_INPUTFORMAT    (0x0014)
```

Description:
It occurs when the format used as the UCBioAPI_INPUT_FIR structure is an unknown format.

■ **UCBioAPIERROR_INVALID_PARAMETER**

Prototype:

```
#define UCBioAPIERROR_INVALID_PARAMETER      (0x0015)
```

Description:
It occurs when a function argument is used or a function argument value over the limit is used.

■ **UCBioAPIERROR_FUNCTION_NOT_SUPPORTED**

Prototype:

```
#define UCBioAPIERROR_FUNCTION_NOT_SUPPORTED (0x0016)
```

Description:
Unsupported function.

5.2.3. Initialization related error definitions

They are definitions on error values related to initialization setting value.
These error values start from the UCBioAPIERROR_BASE_INIT(0x0100) value.

■ **UCBioAPIERROR_INIT_MAXFINGERSFORENROLL**

Prototype:

```
#define UCBioAPIERROR_INIT_MAXFINGERSFORENROLL      (0x0101)
```

Description:

A wrong value was designated at the MaxFingersForEnroll value of UCBioAPI_INIT_INFO_0.

■ **UCBioAPIERROR_INIT_NECESSARYENROLLNUM**

Prototype:

```
#define UCBioAPIERROR_INIT_NECESSARYENROLLNUM       (0x0102)
```

Description:

A wrong value was designated at the NecessaryEnrollNum value of UCBioAPI_INIT_INFO_0.

■ **UCBioAPIERROR_INIT_SAMPLESPERFINGER**

Prototype:

```
#define UCBioAPIERROR_INIT_SAMPLESPERFINGER         (0x0103)
```

Description:

A wrong value was designated at the SamplesPerFinger value of UCBioAPI_INIT_INFO_0.

■ **UCBioAPIERROR_INIT_SECULEVELFORENROLL**
■ **UCBioAPIERROR_INIT_SECULEVELFORVERIFY**
■ **UCBioAPIERROR_INIT_SECULEVELFORIDENTIFY**

Prototype:

```
#define UCBioAPIERROR_INIT_SECULEVELFORENROLL       (0x0104)
#define UCBioAPIERROR_INIT_SECULEVELFORVERIFY       (0x0105)
#define UCBioAPIERROR_INIT_SECULEVELFORIDENTIFY     (0x0106)
```

Description:

A wrong value was designated at the SecurityLevelForEnroll / SecurityLevelForVerify / SecurityLevelForIdentify value of UCBioAPI_INIT_INFO_0.

5.2.4. Device related error definitions

They are definitions on error values related to devices.
These error values start from the UCBioAPIERROR_BASE_DEVICE(0x0200) value.

■ **UCBioAPIERROR_DEVICE_OPEN_FAIL**

Prototype:

```
#define UCBioAPIERROR_DEVICE_OPEN_FAIL       (0x0201)
```

Description:

Device initialization failure. It occurs when a device is not available or the device driver is not installed.

■ **UCBioAPIERROR_INVALID_DEVICE_ID**

Prototype:

```
#define UCBioAPIERROR_INVALID_DEVICE_ID      (0x0202)
```

Description:

It occurs when initializing a device using a wrong device ID.

■ **UCBioAPIERROR_WRONG_DEVICE_ID**

Prototype:

```
#define UCBioAPIERROR_WRONG_DEVICE_ID        (0x0203)
```

Description:

The ID of a device different from the currently initialized device is used.

■ **UCBioAPIERROR_DEVICE_ALREADY_OPENED**

Prototype:

```
#define UCBioAPIERROR_DEVICE_ALREADY_OPENED  (0x0204)
```

Description:

A device is already open. It occurs when initializing a device twice.

■ **UCBioAPIERROR_DEVICE_NOT_OPENED**

Prototype:

```
#define UCBioAPIERROR_DEVICE_NOT_OPENED      (0x0205)
```

Description:

It occurs to try to use a device related function without initializing a device.

■ **UCBioAPIERROR_DEVICE_BRIGHTNESS**

■ **UCBioAPIERROR_DEVICE_CONTRAST**

■ **UCBioAPIERROR_DEVICE_GAIN**

Prototype:

```
#define UCBioAPIERROR_DEVICE_BRIGHTNESS       (0x0206)
#define UCBioAPIERROR_DEVICE_CONTRAST         (0x0207)
#define UCBioAPIERROR_DEVICE_GAIN             (0x0208)
```

Description:

Wrong device setting values for each of brightness / contrast / gain value are used, respectively.

5.2.5. User interface related error definitions

They are definitions on error values related to UI.
These error values start from the UCBioAPIERROR_BASE_UI(0x0300) value.

■ **UCBioAPIERROR_USER_CANCEL**

Prototype:

```
#define UCBioAPIERROR_USER_CANCEL            (0x0301)
```

Description:

A user canceled it by pressing the Cancel button.

■ **UCBioAPIERROR_USER_BACK**

Prototype:

```
#define UCBioAPIERROR_USER_BACK              (0x0302)
```

Description:

A user canceled it by pressing the Back button. Currently, this error value is not used.

■ **UCBioAPIERROR_CAPTURE_TIMEOUT**

Prototype:

```
#define UCBioAPIERROR_CAPTURE_TIMEOUT        (0x0303)
```

Description:

It ends because time is over the limit during fingerprint acquisition.

■ **UCBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS**

Prototype:

```
#define UCBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS       (0x0304)
```

Description:

An acquired fingerprint is suspicious to be a forged fingerprint. Currently, this value is not used.

■ **UCBioAPIERROR_ENROLL_EVENT_PLACE**
■ **UCBioAPIERROR_ENROLL_EVENT_HOLD**
■ **UCBioAPIERROR_ENROLL_EVENT_REMOVE**
■ **UCBioAPIERROR_ENROLL_EVENT_PLACE_AGAIN**

■ **UCBioAPIERROR_ENROLL_EVENT_PROCESS**

■ **UCBioAPIERROR_ENROLL_EVENT_MATCH_FAILED**

Prototype:

```
#define UCBioAPIERROR_ENROLL_EVENT_PLACE          (0x0305)
#define UCBioAPIERROR_ENROLL_EVENT_HOLD           (0x0306)
#define UCBioAPIERROR_ENROLL_EVENT_REMOVE         (0x0307)
#define UCBioAPIERROR_ENROLL_EVENT_PLACE_AGAIN    (0x0308)
#define UCBioAPIERROR_ENROLL_EVENT_PROCESS        (0x0309)
#define UCBioAPIERROR_ENROLL_EVENT_MATCH_FAILED   (0x030a)
```

Description:

When registering the Callback function to FinishCallBackInfo in the UCBioAPI_WINDOW_OPTION structure, it is event values that can passed as dwResult value of the UCBioAPI_WINDOW_CALLBACK_PARAM_1 structure that is the first argument of that function. These values are passed through Callback only during fingerprint registration.

5.2.6. FastSearch related error definitions

They are definitions on error values related to FastSearch.
These error values start from the UCBioAPIERROR_BASE_FASTSEARCH(0x0400) value.

■ **UCBioAPIERROR_FASTSEARCH_INIT_FAIL**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_INIT_FAIL          (0x0401)
```

Description:

FastSearch Engine initialization failure.

■ **UCBioAPIERROR_FASTSEARCH_SAVE_DB**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_SAVE_DB            (0x0402)
```

Description:

Failure of saving of DB file for FastSearch.

■ **UCBioAPIERROR_FASTSEARCH_LOAD_DB**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_LOAD_DB            (0x0403)
```

Description:

Failure of loading of DB file for FastSearch.

■ **UCBioAPIERROR_FASTSEARCH_UNKNOWN_VER**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_UNKNOWN_VER        (0x0404)
```

Description:

Unknown version of DB file for FastSearch.

■ **UCBioAPIERROR_FASTSEARCH_IDENTIFY_FAIL**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_IDENTIFY_FAIL      (0x0405)
```

Description:

Failure of 1:N authentication using FastSearch Engine.

■ **UCBioAPIERROR_FASTSEARCH_DUPLICATED_ID**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_DUPLICATED_ID      (0x0406)
```

Description:

When trying to add FIR to FastSearch DB, a user with the identical ID already exists in DB.

■ **UCBioAPIERROR_FASTSEARCH_IDENTIFY_STOP**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_IDENTIFY_STOP      (0x0407)
```

Description:

1:N authentication is terminated by a user. In general, the Callback function for FastSearch is registered first. During authentication, authentication is terminated if the Callback function returns a value other than 0 and this error value is returned.

■ **UCBioAPIERROR_FASTSEARCH_NOUSER_EXIST**

Prototype:

```
#define UCBioAPIERROR_FASTSEARCH_NOUSER_EXIST       (0x0408)
```

Description:

To find or delete a user in DB for FastSearch, a user with the designated ID does not exist.

5.2.7. Optional value related error definitions

They are definitions on error values related to the UCBioAPI_FIR_OPTIONAL_DATA structure. These error values start from the UCBioAPIERROR_BASE_OPTIONAL(0x0500) value.

- **UCBioAPIERROR_OPTIONAL_UUID_FAIL**
- **UCBioAPIERROR_OPTIONAL_PIN1_FAIL**
- **UCBioAPIERROR_OPTIONAL_PIN2_FAIL**
- **UCBioAPIERROR_OPTIONAL_SITEID_FAIL**
- **UCBioAPIERROR_OPTIONAL_EXPIRE_DATE_FAIL**

Prototype:

```
#define UCBioAPIERROR_OPTIONAL_UUID_FAIL          (0x0501)
#define UCBioAPIERROR_OPTIONAL_PIN1_FAIL          (0x0502)
#define UCBioAPIERROR_OPTIONAL_PIN2_FAIL          (0x0503)
#define UCBioAPIERROR_OPTIONAL_SITEID_FAIL        (0x0504)
#define UCBioAPIERROR_OPTIONAL_EXPIRE_DATE_FAIL   (0x0505)
```

Description:

When performing authentication using the UCBioAPI_VerifyMatchEx function, they are error values upon authentication failure from optional data used as additional authentication data.

UCBioAPIERROR_OPTIONAL_UUID_FAIL:

UUID values of the UCBioAPI_FIR_OPTIONAL_DATA structure do not match each other.

UCBioAPIERROR_OPTIONAL_PIN1_FAIL / UCBioAPIERROR_OPTIONAL_PIN2_FAIL:

Pin1 and pin2 value of the UCBioAPI_FIR_OPTIONAL_DATA structure do not match each other.

UCBioAPIERROR_OPTIONAL_SITEID_FAIL:

SiteID values of the UCBioAPI_FIR_OPTIONAL_DATA structure do not match each other.

UCBioAPIERROR_OPTIONAL_EXPIRED_DATE_FAIL:

The period of ExpireDate of the UCBioAPI_FIR_OPTIONAL_DATA structure is over.

5.2.8. SmartCard related error definitions

They are definitions on error values related to the SmartCard.
These error values start from the UCBioAPIERROR_BASE_SMARTCARD(0x0600) value.

■ **UCBioAPIERROR_SC_FUNCTION_FAILED**

Prototype:

```
#define UCBioAPIERROR_SC_FUNCTION_FAILED            (0x0601)
```

Description:

Smart card function execution failure.

■ **UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE**

Prototype:

```
#define UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE       (0x0602)
```

Description:

Currently, device functions are not supported.

■ **UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE**

Prototype:

```
#define UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE     (0x0603)
```

Description:

A function is not supported because the firmware version of a device is old.

5.3. API References

Definitions on various APIs used in UCBioBSP SDK and function use and arguments are described.

5.3.1. Basic API

APIs used in basic form are described.
These APIs are defined in the UCBioAPI.h file.

■ **UCBioAPI_Init**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_Init (
            [OUT] UCBioAPI_HANDLE_PTR         phHandle);
```

Description:

It is initialized to use UCBioAPI SDK and Handle value is obtained.

Parameters:

phHandle:

The pointer of Handle value to be obtained in UCBioAPI SDK.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_VALIDITY_FAIL
UCBioAPIERROR_EXPIRED_VERSION

■ **UCBioAPI_Terminate**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_Terminate (
            [IN] UCBioAPI_HANDLE              hHandle);
```

Description:

The use of UCBioAPI SDK is terminated and Handle value is closed.

Parameters:

hHandle:

Handle value to be closed in UCBioAPI SDK.

Returns:

UCBioAPIERROR_NONE

UCBioAPIERROR_INVALID_HANDLE

■ **UCBioAPI_GetVersion**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetVersion (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [OUT] UCBioAPI_VERSION_PTR       pVersion);
```

Description:

The version information of UCBioAPI SDK is obtained.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pVersion:

The structure pointer to store the version information.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER

■ **UCBioAPI_GetInitInfo**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetInitInfo (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [IN]  UCBioAPI_UINT8             nStructureType,
            [OUT] UCBioAPI_INIT_INFO_PTR     pInitInfo);
```

Description:

The initialization setting value of UCBioAPI SDK is obtained.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nStructureType:

The type value of the information structure desired to be obtained. This value determines the structure type of pInitInfo. Currently, only 0 is supported.

pInitInfo:

The pointer of the information structure desired to be obtained. At nStructureType, the designated structure must be passed. Currently, only UCBioAPI_INIT_INFO_0 structure is supported but other structures may well be supported later. Before passing the value of StructureType of the pInitInfo as an argument, it must be set to identical to the second argument 'StructureType' to call this function.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_INVALID_TYPE
UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED

■ **UCBioAPI_SetInitInfo**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SetInitInfo (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [IN]  UCBioAPI_UINT8             nStructureType,
            [OUT] UCBioAPI_INIT_INFO_PTR     pInitInfo);
```

Description:

The initial setting value of UCBioAPI SDK is re-designated.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nStructureType:

The type value of the information structure desired to be set. This value determines the structure type of pInitInfo.

pInitInfo:

The pointer of the information structure desired to be set. The structure designated at nStructureType must be passed. Only the UCBioAPI_INIT_INFO_0 structure is currently supported and other structures may be used in the future. Before passing the StructureType value of the pInitInfo structure as an argument, it must be set to be identical to the second argument 'StructureType' value to call this function.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_INVALID_TYPE
UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED
UCBioAPIERROR_INIT_MAXFINGERSFORENROLL
UCBioAPIERROR_INIT_NECESSARYENROLLNUM
UCBioAPIERROR_INIT_SAMPLESPERFINGER
UCBioAPIERROR_INIT_SECULEVELFORENROLL
UCBioAPIERROR_INIT_SECULEVELFORVERIFY
UCBioAPIERROR_INIT_SECULEVELFORIDENTIFY
UCBioAPIERROR_INIT_RESERVED1
UCBioAPIERROR_INIT_RESERVED2

■ **UCBioAPI_SetSkinResource**

Prototype:

```
UCBioAPI_BOOL UCBioAPI UCBioAPI_SetSkinResource (
                    [IN]  LPCTSTR               szResPath);
```

Description:

A skin for the user interface of UCBioAPI SDK is changed.
UCBioAPI SDK uses a skin type UI as the screen used for registration and authentication. Therefore, to use UI in other languages or other forms rather than the standard UI provided by UCBioBSP SDK, a user defined skin can be created and used.

Parameters:

szResPath:
The full path of skin resource DLL to be changed is designated.

Returns:

UCBioAPIERROR_FALSE
UCBioAPIERROR_TRUE

■ **UCBioAPI_SetCaptureQuality**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SetCaptureQuality(
            [IN]  UCBioAPI_HANDLE             hHandle,
            [IN]  UCBioAPI_UINT32             nScanQuality,
            [IN]  UCBioAPI_UINT32             nEnrollQuality,
            [IN]  UCBioAPI_UINT32             nVerifyQuality,
            [IN]  UCBioAPI_UINT32             nIdentifyQuality);
```

Description:

You can set input fingerprint quality values on SDK. Set value ranges from 1 to 100, and value 0(zero) is ignored.(Maintained old value)

Parameters:

nScanQuality : minimum quality value of input finger print image quality.
nEnrollQuality : minimum minutiae quality fo input fingerprint for enroll
nVerifyQuality : minimum minutiae quality fo input fingerprint for 1:1 matching
nIdentifyQuality : minimum minutiae quality fo input fingerprint for 1:N matching

*** now support only nScanQuality, nEnrollQuality.**

Returns: **UCBioAPIERROR_TRUE**

5.3.2. Device related API

APIS related to devices are described.
These APIs are defined in the UCBioAPI.h file.

■ **UCBioAPI_EnumerateDevice**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_EnumerateDevice (
            [IN] UCBioAPI_HANDLE              hHandle,
            [OUT] UCBioAPI_UINT32*            pNumDevice,
            [OUT] UCBioAPI_DEVICE_ID**        ppDeviceID,
            [OUT] UCBioAPI_DEVICE_INFO_EX**   ppDeviceInfoEx);
```

Description:

A list is created after searching all devices installed in the current system.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pNumDevice:

The pointer of the value to be returned with the number of searched devices.

ppDeviceID:

The array pointer to the device ID list. Memory for this pointer array is allocated internally in SDK and it is released automatically at the end of use. Therefore, a function user does not have to allocate or release memory.

ppDeviceInfoEx:

The array pointer of the structure that stores detailed information on devices. Refer to the UCBioAPI_DEVICE_INFO_EX structure. Memory for this pointer array is allocated internally in SDK and it is released automatically at the end of use. Therefore, a function user does not have to allocate or release memory.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_DEVICE_OPEN_FAIL

■ **UCBioAPI_OpenDevice**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_OpenDevice (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [IN]  UCBioAPI_DEVICE_ID   nDeviceID);
```

Description:

A desired device is opened and initialized.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nDeviceID

The ID of a device to be opened. Refer to UCBioAPI_DEVICE_ID.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_DEVICE_ID
UCBioAPIERROR_DEVICE_OPEN_FAIL
UCBioAPIERROR_DEVICE_ALREADY_OPENED

■ **UCBioAPI_CloseDevice**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_CloseDevice (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [IN]  UCBioAPI_DEVICE_ID   nDeviceID);
```

Description:

An opened device is closed and terminated.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nDeviceID

The ID value of a device to be closed. Refer to UCBioAPI_DEVICE_ID.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_WRONG_DEVICE_ID

■ **UCBioAPI_GetDeviceInfo**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetDeviceInfo (
            [IN]  UCBioAPI_HANDLE             hHandle,
            [IN]  UCBioAPI_DEVICE_ID          nDeviceID,
            [IN]  UCBioAPI_UINT8              nStructureType,
            [OUT] UCBioAPI_DEVICE_INFO_PTR    pDeviceInfo);
```

Description:

The device setting value is loaded from a device currently opened.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nDeviceID

The ID value of a device currently opened. Refer to UCBioAPI_DEVICE_ID.

nStructureType:

The type value of the information structure desired to be obtained. This value determines the structure type of pDeviceInfo. This value currently supports only 0.

pDeviceInfo:

The pointer of the information structure desired to be obtained. The structure designated at nStructureType must be passed. Only the UCBioAPI_DEVICE_INIT_INFO_0 structure is currently supported but other structures may be used in the future. Before passing the StructureType value of the pDeviceInfo structure as an argument, it must be set to be identical to the second argument 'StructureType' value to call this function.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_INVALID_TYPE
UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_WRONG_DEVICE_ID

■ **UCBioAPI_SetDeviceInfo**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SetDeviceInfo (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [IN]  UCBioAPI_DEVICE_ID         nDeviceID,
            [IN]  UCBioAPI_UINT8             nStructureType,
            [OUT] UCBioAPI_DEVICE_INFO_PTR   pDeviceInfo);
```

Description:

A new device setting value is re-designated at a currently opened device.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nDeviceID

The device ID value of a currently opened device. Refer to UCBioAPI_DEVICE_I.

nStructureType:

The type value of the information structure desired to be set. This value determines the structure type of pDeviceInfo. This value currently supports only 0.

pDeviceInfo:

The pointer of the information structure desired to be set. The structure designated at nStructureType must be passed. Only the UCBioAPI_DEVICE_INIT_INFO_0 structure is currently supported but other structures may be used in the future. Before passing the StructureType value of the pDeviceInfo structure as an argument, it must be set to be identical to the second argument 'StructureType' value to call this function.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_INVALID_TYPE
UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_WRONG_DEVICE_ID
UCBioAPIERROR_DEVICE_BRIGHTNESS
UCBioAPIERROR_DEVICE_CONTRAST
UCBioAPIERROR_DEVICE_GAIN

■ **UCBioAPI_AdjustDevice**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_AdjustDevice (
            [IN]  UCBioAPI_HANDLE                   hHandle,
            [IN]  const UCBioAPI_WINDOW_OPTION_PTR  pWindowOption);
```

Description:

UI that adjusts the brightness of a currently opened device is launched. But, this function can not be used because currently supported devices internally undergo the automatic brightness adjustment process.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pWindowOption:

The pointer of the UCBioAPI_WINDOW_OPTION structure for UI setting. This value can have NULL. If NULL is set, the default UI setting is used.

For more detailed description, refer to the UCBioAPI_WINDOW_OPTION structure.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_USER_CANCEL

■ **UCBioAPI_GetOpenedDeviceID**

Prototype:

```
UCBioAPI_DEVICE_ID UCBioAPI UCBioAPI_GetOpenedDeviceID (
            [IN]  UCBioAPI_HANDLE      hHandle);
```

Description:

The ID value of a currently opened device is obtained.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

Returns:

UCBioAPI_DEVICE_ID value for the ID of a currently opened device.

■ **UCBioAPI_CheckFinger**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_AdjustDevice (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [OUT] UCBioAPI_BOOL*       pbFingerExist);
```

Description:

It examines if a fingerprint is placed on a currently opened device and the result is notified.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pbFingerExist:

The pointer that stores a value on existence/nonexistence of a fingerprint. The value of either 0 or 1 is stored.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED

■ **UCBioAPI_SetAutoDetect**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SetAutoDetect (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [OUT] UCBioAPI_BOOL        bFlag);
```

Description:

Setting for Auto detect for Finger

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

bFlag:

Flag for auto detect(FALSE:Not Use, TREU:use autodetect ➔ LED OFF)

Returns:

UCBioAPIERROR_NONE

5.3.3. Memory related API

APIs related to FIR data or memory are described.
These APIs are defined in the UCBioAPI.h file.
In general, memory related functions do not require Handle for UCBioBSP SDK as the first argument.

■ **UCBioAPI_GetFIRFromHandle**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetFIRFromHandle (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [OUT] UCBioAPI_FIR_PTR     pFIR);
```

Description:

Real FIR data are obtained from FIR Handle. Since FIR data obtained this way are the structure that includes binary data, they can be saved in file or DB or transmitted over the network after converted into stream data. The memory for FIR data obtained this way must be released using the UCBioAPI_FreeFIR function at the end of use.

Parameters:

hHandle:
A FIR Handle value desired to be obtained.

pFIR:
The pointer of the UCBioAPI_FIR structure desired to be obtained.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER

■ **UCBioAPI_GetExtendedFIRFromHandle**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetExtendedFIRFromHandle (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [OUT] UCBioAPI_VOID_PTR    pFIR,
            [IN]  UCBioAPI_FIR_FORMAT  Format);
```

Description:

Real FIR data are obtained from FIR Handle. It is identical to the UCBioAPI_GetFIRFromHandle function but there is a difference in that it passes format information to obtain the FIR of formats to be added later. The memory for FIR data obtained this way must be released using the UCBioAPI_FreeFIR function at the end of use.

Parameters:

hHandle:

A FIR Handle value desired to be obtained.

pFIR:

The pointer of the UCBioAPI_FIR structure desired to be obtained. Since it is void*, it can respond to other formats later.

Format:

The format value of FIR desired to be obtained. Only UCBioAPI_FIR_FORMAT_STANDARD(0) is currently supported.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_UNKNOWN_FORMAT

■ **UCBioAPI_GetHeaderFromHandle**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetHeaderFromHandle (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [OUT] UCBioAPI_FIR_HEADER_PTR    pHeader);
```

Description:

Only header information of real FIR data is obtained from FIR Handle.

Parameters:

hHandle:

A FIR Handle value desired to be obtained.

pHeader:

The pointer of the UCBioAPI_FIR_HEADER desired to be obtained.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER

■ **UCBioAPI_GetExtendedFIRFromHandle**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetExtendedFIRFromHandle (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [OUT] UCBioAPI_VOID_PTR    pFIR,
            [IN]  UCBioAPI_FIR_FORMAT  Format);
```

Description:

Only header information of real FIR data is obtained from FIR Handle. It is identical to the UCBioAPI_GetHeaderFromHandle function but there is a difference in that it passes format information to obtain the FIR of formats to be added later.

Parameters:

hHandle:

A FIR Handle value desired to be obtained.

pHeader:

The pointer of the UCBioAPI_FIR_HEADER structure desired to be obtained. Since it is void*, it can respond to other formats later.

Format:

The format value of FIR desired to be obtained. Only UCBioAPI_FIR_FORMAT_STANDARD(0) is currently supported.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_UNKNOWN_FORMAT

■ **UCBioAPI_GetTextFIRFromHandle**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetTextFIRFromHandle (
            [IN]  UCBioAPI_HANDLE             hHandle,
            [OUT] UCBioAPI_FIR_TEXTENCODE_PTR pTextFIR,
            [IN]  UCBioAPI_BOOL               bIsWide);
```

Description:

Real FIR data are obtained in text string type from FIR Handle. Since FIR data obtained this way are the structure that includes string type data, they can be saved in file or DB or transmitted over the network after converted to stream data. The memory for FIR data saved this way must be released using the UCBioAPI_FreeTextFIR function at the end of use.

Parameters:

hHandle:

A FIR Handle value desired to be obtained.

pTextFIR:

The pointer of the UCBioAPI_FIR_TEXTENCODE structure desired to be obtained.

bIsWide:

It designates if a text string desired to be obtained should be in Unicode. If UCBioAPI_TRUE is set, a text string to be obtained is returned in Unicode type.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER

■ **UCBioAPI_GetExtendedTextFIRFromHandle**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetExtendedTextFIRFromHandle (
            [IN]  UCBioAPI_HANDLE             hHandle,
            [OUT] UCBioAPI_FIR_TEXTENCODE_PTR pTextFIR,
            [IN]  UCBioAPI_BOOL               bIsWide,
            [IN]  UCBioAPI_FIR_FORMAT         Format);
```

Description:

Real FIR data are obtained from FIR Handle. It is identical to the UCBioAPI_GetFIRFromHandle function but there is a difference in that it passes format information to obtain the FIR of formats to be added later. The memory for FIR obtained this way must be released using the UCBioAPI_FreeFIR at the end of use.

Parameters:

hHandle:

A FIR Handle value desired to be obtained.

pTextFIR:

The pointer of the UCBioAPI_FIR_TEXTENCODE structure desired to be obtained.

bIsWide:

It designates if a text string desired to be obtained should be in Unicode. If UCBioAPI_TRUE is set, a text string to be obtained is returned in Unicode type.

Format:

The format value of FIR desired to be obtained. Only UCBioAPI_FIR_FORMAT_STANDARD(0) is currently supported.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_UNKNOWN_FORMAT

■ **UCBioAPI_FreeFIRHandle**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FreeFIRHandle (
                              [IN]  UCBioAPI_HANDLE  hHandle);
```

Description:

FIR Handle is terminated and memory is released. If the value of FIR Handle is UCBioAPI_INVALID_HANDLE, nothing is done.

Parameters:

hHandle:

The value of FIR Handle to be terminated.

Returns:

UCBioAPIERROR_NONE

UCBioAPIERROR_INVALID_HANDLE

■ **UCBioAPI_FreeFIR**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FreeFIR (
            [IN]  UCBioAPI_VOID_PTR    pFIR);
```

Description:

The use of FIR data is terminated and memory is released. If the pFIR value is NULL, nothing is done.

Parameters:

pFIR:

The pointer of the FIR structure to be terminated. To support FIR formats to be added later, it is set to void*.

Returns:

UCBioAPIERROR_NONE

UCBioAPIERROR_UNKNOWN_FORMAT

■ **UCBioAPI_FreeTextFIR**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FreeTextFIR (
            [IN]  UCBioAPI_FIR_TEXTENCODE_PTR       pTextFIR);
```

Description:

The use of text FIR data is terminated and memory is released. If the pTextFIR value is NULL, nothing is done.

Parameters:

pTextFIR:

The pointer of the text FIR structure to be terminated.

Returns:

UCBioAPIERROR_NONE

■ **UCBioAPI_FreePayload**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FreePayload (
            [IN]  UCBioAPI_FIR_PAYLOAD_PTR          pPayload);
```

Description:

The use of Payload data is terminated and memory is released. If the Payload value is NULL, nothing is done.

Parameters:

pPayload:

The structure of the Payload structure to be terminated.

Returns:

UCBioAPIERROR_NONE

5.3.4. Core API

Core APIs, UCBioBSP SDK's key functions, are described.
These APIs are defined in the UCBioAPI.h file.

■ **UCBioAPI_Capture**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_Capture (
              [IN]  UCBioAPI_HANDLE                   hHandle,
              [IN]  UCBioAPI_FIR_PURPOSE              nPurpose,
              [OUT] UCBioAPI_FIR_HANDLE_PTR           phCapturedFIR,
              [IN]  UCBioAPI_SINT32                   nTimeout,
              [OUT] UCBioAPI_FIR_HANDLE_PTR           phAuditData,
              [IN]  const UCBioAPI_WINDOW_OPTION_PTR pWindowOption);
```

Description:

FIR Handle is created by acquiring 1 fingerprint from a currently opened device. The memory for phCapturedFIR or phAuditData obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use. Since this function uses a device, a device must be opened before use.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nPurpose:

The purpose value of FIR data can be designated. Allowed values are shown below.

```
#define UCBioAPI_FIR_PURPOSE_VERIFY                         (0x01)
#define UCBioAPI_FIR_PURPOSE_IDENTIFY                       (0x02)
#define UCBioAPI_FIR_PURPOSE_ENROLL                         (0x03)
#define UCBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY   (0x04)
#define UCBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY (0x05)
#define UCBioAPI_FIR_PURPOSE_AUDIT                          (0x06)
#define UCBioAPI_FIR_PURPOSE_UPDATE                         (0x10)
```

Each of values is used only as reference for FIR data and it does not have any effect on authentication. If this value is set as Enroll related purpose, it operates in the same way as the UCBioAPI_Enroll function.
For more detailed description, refer to the UCBioAPI_FIR_PURPOSE reference.

phCapturedFIR:
The pointer to store the FIR Handle value of acquired fingerprint data.

nTimeout:
The maximum waiting time for fingerprint acquisition. As the unit is millisecond, 1,000 is set for 1 second. The following additional values are allowed.

```
#define UCBioAPI_NO_TIMEOUT                 (0)
#define UCBioAPI_USE_DEFAULT_TIMEOUT        (-1)
#define UCBioAPI_CONTINUOUS_CAPTRUE         (-2)
```

If 0 is set, waiting for fingerprint input continues without timeout. If a fingerprint is entered while waiting, input process is terminated and it is returned.
- If 1 is set, it waits as much as the default waiting time designated in the UCBioAPI_INIT_INFO structure.
- If 2 is set, it is not terminated even if a fingerprint is entered and it continues to accept fingerprint input.

phAuditData:
The pointer to store the FIR Handle value of acquired fingerprint image data. This value can not have NULL value. If NULL is set, no value is returned.

pWindowOption:
The pointer of the UCBioAPI_WINDOW_OPTION structure for UI setting. This value can have NULL value. If NULL is set, the default UI setting is used.
For more detailed description, refer to the UCBioAPI_WINDOW_OPTION structure.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_USER_CANCEL
UCBioAPIERROR_CAPTURE_TIMEOUT
UCBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS

■ **UCBioAPI_Process**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_Process (
              [IN]  UCBioAPI_HANDLE                 hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR    piCapturedFIR,
              [OUT] UCBioAPI_FIR_HANDLE_PTR         phProcessedFIR);
```

Description:

FIR Handle is created by extracting fingerprint special features from Audit FIR. The memory for phProcessedFIR obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

piCapturedFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data for extracting special features. Refer to the UCBioAPI_INPUT_FIR structure.

phProcessedFIR:

The pointer to store the FIR Handle value of acquired fingerprint data.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_UNKNOWN_INPUTFORMAT
UCBioAPIERROR_ALREADY_PROCESSED
UCBioAPIERROR_DATA_PROCESS_FAIL

■ **UCBioAPI_CreateTemplate**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_CreateTemplate (
              [IN]  UCBioAPI_HANDLE                  hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piCapturedFIR,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piStoredFIR,
              [OUT] UCBioAPI_FIR_HANDLE_PTR          phNewFIR,
              [IN]  const UCBioAPI_FIR_PAYLOAD_PTR   pPayload);
```

Description:

A new FIR Handle is created by merging new FIR data with existing FIR data or replacing with new FIR data. Also, it can be used in replacing existing FIR with new Payload. The memory for phNewFIR obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

piCapturedFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data to be used in replacement. Refer to the UCBioAPI_INPUT_FIR structure.

piStoredFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data to be used as a base. Refer to the UCBioAPI_INPUT_FIR structure.

If this value is not NULL, FIR data are created using this value as a base.

Assume that this FIR value stores data for thumb, index and middle of the right hand and piCapturedFIR data, the new replacement, store data for the right hand thumb and left hand thumb. The final FIR data to be created will include all - right hand index and middle in piStoredFIR data and right hand thumb and left hand thumb in piCapturedFIR data. That is, new data are created from existing data through the addition and modification of new data.

phNewFIR:

The pointer to store the FIR Handle value of acquired fingerprint data.

pPayload:

The pointer of the structure that stores Payload data to be stored at phNewFIR data to be created. This value can use NULL. If NULL is set, the Payload value of existing data is kept.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_MUST_BE_PROCESSED_DATA
UCBioAPIERROR_EXTRACTION_OPEN_FAIL
UCBioAPIERROR_UNKNOWN_FORMAT

■ **UCBioAPI_VerifyMatch**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_VerifyMatch (
              [IN]  UCBioAPI_HANDLE                  hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piProcessedFIR,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piStoredFIR,
              [OUT] UCBioAPI_BOOL*                   pbResult,
              [IN]  const UCBioAPI_FIR_PAYLOAD_PTR   pPayload);
```

Description:

Previously acquired two FIR data are compared and that authentication result is obtained. If authentication succeeds, the Payload value stored in FIR data for registration can also be obtained.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

piProcessedFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data requiring for authentication. Refer to the UCBioAPI_INPUT_FIR structure.

piStoredFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data for registration requiring authentication. Refer to the UCBioAPI_INPUT_FIR structure.

pbResult:

The pointer to store the authentication result value. The value of either 0 or 1 is returned.

pPayload:

When authentication succeeds, the Payload value stored in piStoredFIR can be obtained and the pointer of the structure to be returned with that value is designated. This value can use NULL. If NULL is set, the Payload value is not returned.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_MUST_BE_PROCESSED_DATA
UCBioAPIERROR_EXTRACTION_OPEN_FAIL
UCBioAPIERROR_UNKNOWN_FORMAT

■ **UCBioAPI_VerifyMatchEx**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_VerifyMatchEx (
              [IN]  UCBioAPI_HANDLE                  hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piProcessedFIR,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piStoredFIR,
              [OUT] UCBioAPI_BOOL*                   pbResult,
              [IN]  const UCBioAPI_FIR_PAYLOAD_PTR   pPayload,
              [IN]  UCBioAPI_MATCH_OPTION_PTR        pMatchOption);
```

Description:

Previously acquired two FIR data are compared and that authentication result is obtained. When authentication succeeds, the Payload value stored in FIR data for registration can be obtained.

It is identical to the UCBioAPI_VerifyMatch function and also it can designate values to use additional authentication data for authentication. If these values are designated, fingerprint authentication is performed only when authentication for these values succeeds before fingerprint authentication.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

piProcessedFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data requiring authentication. Refer to the UCBioAPI_INPUT_FIR structure.

piStoredFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data for registration requiring authentication. Refer to the UCBioAPI_INPUT_FIR structure.

pbResult:

The pointer to store the authentication result value. The value of either 0 or 1 is returned.

pPayload:

When authentication succeeds, the Payload value stored in piStoredFIR can be obtained and it designates the pointer of the structure to be returned with that value. This value can use NULL. If NULL is set, the Payload value is not returned.

pMatchOption:

Values to use additional authentication data for authentication are designated. For detailed

descriptions on these values, refer to the UCBioAPI_MATCH_OPTION structure.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_MUST_BE_PROCESSED_DATA
UCBioAPIERROR_EXTRACTION_OPEN_FAIL
UCBioAPIERROR_UNKNOWN_FORMAT
UCBioAPIERROR_OPTIONAL_UUID_FAIL
UCBioAPIERROR_OPTIONAL_PIN1_FAIL
UCBioAPIERROR_OPTIONAL_PIN2_FAIL
UCBioAPIERROR_OPTIONAL_SITEID_FAIL
UCBioAPIERROR_OPTIONAL_EXPIRE_DATE_FAIL

■ **UCBioAPI_Enroll**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_Enroll (
              [IN]  UCBioAPI_HANDLE                  hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piStoredFIR,
              [OUT] UCBioAPI_FIR_HANDLE_PTR          phEnrolledFIR,
              [IN]  const UCBioAPI_FIR_PAYLOAD_PTR   pPayload,
              [IN]  UCBioAPI_SINT32                  nTimeout,
              [OUT] UCBioAPI_FIR_HANDLE_PTR          phAuditData,
              [IN]  const UCBioAPI_WINDOW_OPTION_PTR pWindowOption);
```

Description:

FIR Handle is created by acquiring fingerprints of 1 person from a currently opened device. Since they are fingerprints of 1 person, fingerprints for several fingers are registered at the same time to create one FIR data. The memory for phEnrolledFIR or phAuditData obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use. Since this function uses a device, a device must be opened before use.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

piStoredFIR:

To modify existing FIR data, this value is designated. If this value is set as NULL, a new fingerprint is entered.

phEnrolledFIR:

The pointer to store the FIR Handle value of registered fingerprint data.

pPayload:

The pointer of the structure that stores Payload data to be stored at phNewFIR data to be created. This value can use NULL. If NULL is set, the Payload value of existing data is kept.

nTimeout:

The maximum waiting time for fingerprint acquisition. Since the unit is millisecond, 1,000 represents 1 second. The following additional values are allowed.

```
#define UCBioAPI_NO_TIMEOUT                (0)
#define UCBioAPI_USE_DEFAULT_TIMEOUT       (-1)
```

If 0 is set, it continues to wait for fingerprint input without timeout. If a fingerprint is

entered while waiting, input process is terminated and it proceeds to the next step.
-If 1 is set, it waits as much as the default waiting time designated in the UCBioAPI_INIT_INFO structure.
-2 is not used in the UCbioAPI_Enroll function.

phAuditData:
The pointer to store the FIR Handle value of acquired fingerprint image data. This value can have NULL. If NULL is set, the value is not returned.

pWindowOption:
The pointer of the UCBioAPI_WINDOW_OPTION structure for UI setting. This value can have NULL. If NULL is set, the default UI setting is used.
For more detailed description, refer to the UCBioAPI_WINDOW_OPTION structure.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FUNCTION_FAIL
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_USER_CANCEL
UCBioAPIERROR_USER_BACK

■ **UCBioAPI_Verify**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_Verify (
              [IN]  UCBioAPI_HANDLE                   hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR      piStoredFIR,
              [OUT] UCBioAPI_BOOL*                    pbResult,
              [OUT] UCBioAPI_FIR_PAYLOAD_PTR          pPayload,
              [IN]  UCBioAPI_SINT32                   nTimeout,
              [OUT] UCBioAPI_FIR_HANDLE_PTR           phAuditData,
              [IN]  const UCBioAPI_WINDOW_OPTION_PTR pWindowOption);
```

Description:

Previously acquired FIR data and fingerprint data entered in real time from the current device are compared and the authentication result is obtained. If authentication succeeds, the Payload value stored in FIR data for registration can be also obtained. It is nearly identical to the UCBioAPI_VerifyMatch function but there is a difference in that it performs authentication by accepting a fingerprint in real time.

That is, it can be thought that both the UCBioAPI_Capture and UCBioAPI_VerifyMatch functions are internally executed together. Since this function uses a device, a device must be opened before use.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

piStoredFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data for registration requiring authentication. Refer to the UCBioAPI_INPUT_FIR structure.

pbResult:

The pointer to store the authentication result value. The value of either 0 or 1 is returned.

pPayload:

When authentication succeeds, the Payload value stored in piStoredFIR can be obtained, and the pointer of the structure to be returned with that value is designated. This value can use NULL. If NULL is set, the Payload value is not returned.

nTimeout:

The maximum waiting time for fingerprint registration. Since the unit is millisecond, 1,000 represents 1 second. The following additional values are allowed.

```
#define UCBioAPI_NO_TIMEOUT                    (0)
#define UCBioAPI_USE_DEFAULT_TIMEOUT           (-1)
#define UCBioAPI_CONTINUOUS_CAPTRUE            (-2)
```

If 0 is set, it continues to wait without timeout. If a fingerprint is entered while waiting, input process is terminated and it is returned.
-If 1 is set, it waits as much as the default waiting time designated in the UCBioAPI_INIT_INFO structure.
-If 2 is set, it continues to accept fingerprint input even if a fingerprint is entered.

phAuditData:
The pointer to store the FIR Handle value of acquired fingerprint image data. This value can have NULL. If NULL is set, a value is not returned.

pWindowOption:
The pointer of the UCBioAPI_WINDOW_OPTION structure for UI setting. This value can have NULL. If NULL is set, the default UI setting is used.
For more detailed description, refer to the UCBioAPI_WINDOW_OPTION structure.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_MUST_BE_PROCESSED_DATA
UCBioAPIERROR_EXTRACTION_OPEN_FAIL
UCBioAPIERROR_UNKNOWN_FORMAT
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_USER_CANCEL
UCBioAPIERROR_CAPTURE_TIMEOUT
UCBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS

5.3.5. Data conversion API

APIs related to data conversion are described.
These APIs are defined in the UCBioAPI_Export.h file.

■ **UCBioAPI_FIRToTemplate**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FIRToTemplate (
              [IN]  UCBioAPI_HANDLE                  hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR     piFIR,
              [OUT] UCBioAPI_EXPORT_DATA_PTR         pExportData,
              [IN]  UCBioAPI_TEMPLATE_TYPE           nExportType);
```

Description:

Template information in desired type is obtained from FIR data. After calling the function, the structure that stores each of template informations can be obtained. The memory for pExportData obtained this way must be released using the UCBioAPI_FreeExportData function at the end of use.
Because FIR data generally hide internal data through encryption, template-by-template information is not known. To process data template-by-template as in other application programs or in the terminal, data need to be converted using this function before use.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

piFIR:
The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data requiring conversion. Refer to the UCBioAPI_INPUT_FIR structure.

pExportData:
The pointer of the structure to store converted template data. Refer to the UCBioAPI_EXPORT_DATA structure.

nExportType:
Template data type to convert is designated.
For information of allowed values, refer to the UCBioAPI_TEMPLATE_TYPE definition.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE

UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_MUST_BE_PROCESSED_DATA
UCBioAPIERROR_FUNCTION_FAIL

■ **UCBioAPI_TemplateToFIR**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_TemplateToFIR (
              [IN]  UCBioAPI_HANDLE           hHandle,
              [IN]  UCBioAPI_UINT8*           pTemplateData,
              [IN]  UCBioAPI_UINT32           nTemplateDataSize,
              [IN]  UCBioAPI_UINT32           nTemplateDataType,
              [IN]  UCBioAPI_FIR_PURPOSE      nPurpose,
              [OUT] UCBioAPI_FIR_HANDLE_PTR   phProcessedFIR);
```

Description:

FIR Handle is created using 1 template information in a specific type. After calling the function, FIR Handle with template information can be obtained. The memory for phProcessedFIR obtained this way must be released using the UCBioAPI_FreeFIRHandle at the end of use. In general, UCBioBSP SDK does not allow direct authentication using template and all data are required to be converted into FIR type before use. Therefore, it is necessary to convert all data into FIR data using this function.

This function is used to convert template data into FIR Handle simply. To convert template data into FIR Handle with more detailed information, use the UCBioAPI_ImportDataToFIR function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pTemplateData:

The memory block that stores 1 template data requiring conversion.

nTemplateDataSize:

The length of template data to be converted.

nTemplateDataType:

The type of template data to be converted is designated.

For information on allowed values, refer to the UCBioAPI_TEMPLATE_TYPE definition.

nPurpose:

The purpose value of FIR data to convert can be designated.

These values are used only for reference on FIR data and they have no effect on authentication. For more detailed description, refer to the UCBioAPI_FIR_PURPOSE definition.

phProcessedFIR:
The pointer to store the FIR Handle value of converted fingerprint data.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FUNCTION_FAIL
UCBioAPIERROR_INVALID_TEMPLATESIZE

■ **UCBioAPI_TemplateToFIREx**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_TemplateToFIREx (
              [IN]  UCBioAPI_HANDLE          hHandle,
              [IN]  UCBioAPI_UINT8*          pTemplateData,
              [IN]  UCBioAPI_UINT32          nTemplateDataSize,
              [IN]  UCBioAPI_UINT32          nOneTemplateDataSize,
              [IN]  UCBioAPI_UINT32          nTemplateDataType,
              [IN]  UCBioAPI_FIR_PURPOSE     nPurpose,
              [OUT] UCBioAPI_FIR_HANDLE_PTR  phProcessedFIR);
```

Description:

FIR Handle is created using a large number of template informations in a specific type. After calling the function, FIR Handle with template information can be obtained. The memory for phProcessedFIR obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use.

In general, UCBioBSP SDK does not allow direct authentication using template and all data are required to be converted into FIR type before use. Therefore, it is necessary to convert all data into FIR data using this function before use.

This function is used to simply convert data into FIR Handle. To convert data into FIR Handle with more detailed information, use the UCBioAPI_ImportDataToFIR function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pTemplateData:

The memory block that stores a large number of template data requiring conversion.

nTemplateDataSize:

The entire length of template data to be converted.

The entire data length must be a multiple of nOneTemplateDataSize. To create FIR using two of template data with 400 byte size, make pTemplateData an 800 byte size memory block and store two data in succession. After that, designate nTemplateDataSize with 800 and nOneTemplateDataSize with 400.

nOneTemplateDataSize:

The length of 1 template data to be converted.

nTemplateDataType:

The type of template data to be converted is designated.

For information on allowed values, refer to the UCBioAPI_TEMPLATE_TYPE definition.

nPurpose:
The purpose value of FIR data to convert can be designated.
These values are used only for reference on FIR data and they have no effect on authentication. For more detailed description, refer to the UCBioAPI_FIR_PURPOSE definition.

phProcessedFIR:
The pointer to store the FIR Handle of converted fingerprint data.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FUNCTION_FAIL
UCBioAPIERROR_INVALID_TEMPLATESIZE

■ **UCBioAPI_ConvertTemplateData**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_ConvertTemplateData (
              [IN]  UCBioAPI_HANDLE          hHandle,
              [IN]  UCBioAPI_UINT8*          pTemplateData,
              [IN]  UCBioAPI_UINT32          nTemplateDataSize,
              [IN]  UCBioAPI_UINT32          nOneTemplateDataSize,
              [IN]  UCBioAPI_UINT32          nTemplateDataType,
              [IN]  UCBioAPI_UINT32          nConvertType,
              [OUT] UCBioAPI_UINT8**         ppConvertedData,
              [OUT] UCBioAPI_UINT32**        ppConvertedDataLen);
```

Description:

This function is used when converting into another type of template information using a large number of template informations in a specific type. After calling the function, the memory block point with template information can be obtained. The memory for ppConvertedData and ppConvertedDataLen obtained this way must be released using the UCBioAPI_FreeData function at the end of use.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pTemplateData:

The memory block that stores a large number of template data requiring conversion.

nTemplateDataSize:

The entire length of template data to be converted.

The entire data length must be a multiple of nOneTemplateDataSize. To create FIR using two of template date with 400 byte size, make pTemplateData 800 byte size memory block and store two data in succession. After that, designate nTemplateDataSize with 800 and nOneTemplateDataSize with 400.

nOneTemplateDataSize:

The length of 1 template data to be converted.

nTemplateDataType:

The type of template data to be converted is designated.

For information on allowed values, refer to the UCBioAPI_TEMPLATE_TYPE definition.

nConvertType:

The type of template data to convert is designated.
For information on allowed values, refer to the UCBioAPI_TEMPLATE_TYPE definition.

ppConvertedData:
The memory block pointer to store template data to convert. Since memory for this value is allocated internally, the memory must be released using the UCBioAPI_FreeData function at the end of use.
When a large number of templates were converted, a large number of templates are stored sequentially in this memory block. Each of template size is stored sequentially at ppConvertedDataLen as array.

ppConvertedDataLen:
The memory block pointer to store the length of template data to convert. Since memory for this value is allocated internally, the memory must be released using the UCBioAPI_FreeData function at the end of use.
This value is an array type and length values are stored sequentially when a large number of templates were converted.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FUNCTION_FAIL
UCBioAPIERROR_INVALID_TEMPLATESIZE

■ **UCBioAPI_ImportDataToFIR**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_ImportDataToFIR (
              [IN]  UCBioAPI_HANDLE           hHandle,
              [IN]  UCBioAPI_EXPORT_DATA_PTR  pExportData,
              [IN]  UCBioAPI_FIR_PURPOSE      nPurpose,
              [OUT] UCBioAPI_FIR_HANDLE_PTR   phProcessedFIR);
```

Description:

FIR Handle is created using a large number of template informations in a specific type. After calling the function, FIR Handle with template information can be obtained. The memory for phProcessedFIR obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use.

In general, UCBioBSP SDK does not allow direct authentication using template and all data are required to be converted into FIR type before use. Therefore, it is necessary to convert data into FIR data using this function.

This function implements the same thing as the UCBioAPI_TemplateToFIR function but it can convert data into FIR Handle while including more detailed finger information.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pExportData:

The pointer that stores template data to be converted. Refer to the UCBioAPI_EXPORT_DATA structure.

nPurpose:

The purpose value of FIR data to convert can be designated.

These values are used only for reference on FIR data and they do not have any effect on authentication. For more detailed description, refer to the UCBioAPI_FIR_PURPOSE definition.

phProcessedFIR:

The pointer to store the FIR Handle value of converted fingerprint data.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FUNCTION_FAIL

UCBioAPIERROR_INVALID_TEMPLATESIZE

■ **UCBioAPI_ImportDataToFIREx**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_ImportDataToFIREx (
              [IN]  UCBioAPI_HANDLE           hHandle,
              [IN]  UCBioAPI_EXPORT_DATA_PTR  pExportData,
              [IN]  UCBioAPI_FIR_PURPOSE      nPurpose,
              [IN]  UCBioAPI_FIR_DATA_TYPE    nDataType,
              [OUT] UCBioAPI_FIR_HANDLE_PTR   phProcessedFIR,
              [IN]  UCBioAPI_VOID_PTR         pReserved);
```

Description:

FIR Handle is created using a large number of templates in a specific type. After calling the function, FIR Handle with template information can be obtained. The memory for phProcessedFIR obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use.

In general, UCBioBSP SDK does not allow direct authentication and all data are required to be converted into FIR type before use. Therefore, it is necessary to convert data into FIR data using this function.

This function implements the same operation as the UCBioAPI_ImportDataToFIR function but it can designate the data type information of FIR also.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pExportData:

The pointer that stores template data to be converted. Refer to the UCBioAPI_EXPORT_DATA structure.

nPurpose:

The purpose value of FIR data to convert can be designated.

These values are used only for reference on FIR data and they do not have any effect on authentication. For more detailed description, refer to the UCBioAPI_FIR_PURPOSE definition.

nDataType:

The data type of FIR data to convert can be designated.

For more detailed description on these values, refer to the UCBioAPI_FIR_DATA_TYPE definition.

phProcessedFIR:

The pointer to store the FIR Handle value of converted fingerprint data.

pReserved:
Reserved argument.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FUNCTION_FAIL
UCBioAPIERROR_INVALID_TEMPLATESIZE

■ **UCBioAPI_AuditFIRToImage**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_AuditFIRToImage (
              [IN]  UCBioAPI_HANDLE                 hHandle,
              [IN]  const UCBioAPI_INPUT_FIR_PTR    piAuditFIR,
              [OUT] UCBioAPI_EXPORT_AUDIT_DATA_PTR  pExportAuditData);
```

Description:

Image information in a desired raw format is obtained from Audit FIR data that stores image information. After calling the function, the structure with each of image informations can be obtained. The memory for pExportAuditData obtained this way must be released using the UCBioAPI_FreeExportAuditData function at the end of use.

Since Audit FIR data generally hide internal data through encryption, finger-by-finger image information can not be known. Therefore, to process data in finger-by-finger as in other application programs or in the terminal, data need to be converted using this function to use image stored in Audit FIR.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

piAuditFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores Audit FIR data requiring conversion. Refer to the UCBioAPI_INPUT_FIR structure.

pExportAuditData:

The pointer to store converted image data. Refer to the UCBioAPI_EXPORT_AUDIT_DATA structure.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_ALREADY_PROCESSED
UCBioAPIERROR_UNKNOWN_INPUTFORMAT
UCBioAPIERROR_FUNCTION_FAIL

■ **UCBioAPI_ImageToAuditFIR**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_ImportDataToFIR (
              [IN]  UCBioAPI_HANDLE                 hHandle,
              [IN]  UCBioAPI_EXPORT_AUDIT_DATA_PTR  pExportAuditData,
              [OUT] UCBioAPI_FIR_HANDLE_PTR         phAuditFIR);
```

Description:

Audit FIR Handle is created using a large number of image informations in the raw format. After calling the function, Audit FIR Handle with image information can be obtained. The memory for phAuditFIR obtained this way must be released using the UCBioAPI_FreeFIRHandle function at the end of use.

In general, UCBioBSP SDK does not allow direct authentication using image and all data need to be converted into FIR type before use. Therefore, it is necessary to convert data into FIR data using this function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pExportAuditData:

The pointer of the structure that stores image data to be converted. Refer to the UCBioAPI_EXPORT_AUDIT_DATA structure.

phAuditFIR:

The pointer to store the FIR Handle value of converted fingerprint image data.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FUNCTION_FAIL

■ **UCBioAPI_FreeData**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FreeData (
              [IN] UCBioAPI_UINT8*            pData);
```

Description:

The memory of data created by the UCBioAPI_ConvertTemplateData function is released.

Parameters:

pData:

The pointer of the block data to release. If this value is NULL, nothing is done.

Returns:

UCBioAPIERROR_NONE

■ **UCBioAPI_FreeExportData**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FreeExportData (
              [IN] UCBioAPI_EXPORT_DATA_PTR   pExportData);
```

Description:

The memory of the UCBioAPI_EXPORT_DATA structure created by the UCBioAPI_FIRToTemplate function is released.

Parameters:

pExportData:

The pointer of the UCBioAPI_EXPORT_DATA structure to release. If this value is NULL, nothing is done.

Returns:

UCBioAPIERROR_NONE

■ **UCBioAPI_FreeExportAuditData**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_FreeExportAuditData (
              [IN] UCBioAPI_EXPORT_AUDIT_DATA_PTR   pExportAuditData);
```

Description:

The memory of the UCBioAPI_EXPORT_AUDIT_DATA structure created by the UCBioAPI_AuditFIRToImage function is released.

Parameters:

pExportAuditData:

The pointer of the UCBioAPI_EXPORT_AUDIT_DATA structure to release. If this value is NULL, nothing is done.

Returns:

UCBioAPIERROR_NONE

5.3.6. FastSearch API

APIs related to FastSearch are described.
These APIs are defined in the UCBioAPI_FastSearch.h file.

■ **UCBioAPI_InitFastSearchEngine**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_InitFastSearchEngine (
              [IN]  UCBioAPI_HANDLE    hHandle);
```

Description:

FastSearch Engine is initialized.
To use FastSearch Engine, it must be initialized using this function. At the end of use, it must be terminated using the UCBioAPI_TerminateFastSearchEngine function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_VERIFICATION_OPEN_FAIL

■ **UCBioAPI_TerminateFastSearchEngine**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_TerminateFastSearchEngine (
              [IN]  UCBioAPI_HANDLE    hHandle);
```

Description:

The use of FastSearch Engine is terminated.
At the end of FastSearch Engine use, it must be terminated using this function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FASTSEARCH_INIT_FAIL

■ **UCBioAPI_GetFastSearchInitInfo**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetFastSearchInitInfo (
              [IN]  UCBioAPI_HANDLE           hHandle,
              [IN]  UCBioAPI_UINT8            nStructureType,
              [OUT] UCBioAPI_INIT_INFO_PTR    pInitInfo);
```

Description:

The initial setting value of FastSearch Engine is obtained.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nStructureType:

The type value of the information structure desired to be obtained. This value determines the structure type of pInitInfo. Currently, 0 is supported.

pInitInfo:

The pointer of the information structure desired to be obtained. The structure designated at nStructureType must be passed. Only the UCBioAPI_FASTSEARCH_INIT_INFO_0 is currently supported but other structures may be used in the future. Before passing the StructureType value of the pInitInfo structure as an argument, it must be set to be identical to the second argument 'StructureType' value to call this function.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_INVALID_TYPE
UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED
UCBioAPIERROR_FASTSEARCH_INIT_FAIL

■ **UCBioAPI_SetFastSearchInitInfo**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SetFastSearchInitInfo (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [IN]  UCBioAPI_UINT8             nStructureType,
            [OUT] UCBioAPI_INIT_INFO_PTR     pInitInfo);
```

Description:

The initial setting value of FastSearch Engine is re-designated.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nStructureType:

The type value of the information structure desired to be set. This value determines the structure type of pInitInfo. Currently, only 0 is supported.

pInitInfo:

The pointer of the information structure desired to be set. The structure designated at nStructureType must be passed. Only the UCBioAPI_FASTSEARCH_INIT_INFO_0 structure is currently supported but other structures may be used in the future. Before passing the StructureType value of the pInitInfo structure as an argument, it must be set to be identical to the second argument 'StructureType' value to call this function.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_INVALID_TYPE
UCBioAPIERROR_STRUCTTYPE_NOT_MATCHED
UCBioAPIERROR_FASTSEARCH_INIT_FAIL

■ **UCBioAPI_AddFIRToFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_AddFIRToFastSearchDB (
            [IN]  UCBioAPI_HANDLE                    hHandle,
            [IN]  const UCBioAPI_INPUT_FIR_PTR       pInputFIR,
            [IN]  UCBioAPI_UINT32                    nUserID,
            [OUT] UCBioAPI_FASTSEARCH_SAMPLE_INFO_PTR pSampleInfo);
```

Description:

FIR data are added to DB for FastSearch.

For 1:N authentication, the DB to implement 1:N authentication needs to be built first. That DB is created using this function.

Also, 1:N authentication internally operates in template unit not in FIR unit. Therefore, even if one FIR is added, several numbers of data are added to the DB when several numbers of templates exist inside FIR. Information in added templates can be obtained through the pSampleInfo value.

The memory for the DB created this way must be released using the UCBioAPI_ClearFastSearchDB function at the end of use.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pInputFIR:

The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data to be added.
Refer to the UCBioAPI_INPUT_FIR structure.

nUserID:

The user ID of FIR to be added.

pSampleInfo:

The value on template information of FIR added to the DB can be obtained. If it is not necessary to obtain this value, NULL is set. Refer to the UCBioAPI_FASTSEARCH_SAMPLE_INFO structure.

Using this value, data on templates added by a user application program can be managed. For detailed example of use, refer to the UCBioBSP_FastSearchDemo folder in the Samples folder in SDK.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE

UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL
UCBioAPIERROR_FASTSEARCH_DUPLICATED_ID
UCBioAPIERROR_FASTSEARCH_UNKNOWN_VER

■ **UCBioAPI_RemoveFpFromFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_RemoveFpFromFastSearchDB (
            [IN]  UCBioAPI_HANDLE                   hHandle,
            [IN]  UCBioAPI_FASTSEARCH_FP_INFO_PTR   pFpInfo);
```

Description:

Specific data are deleted from the DB for FastSearch.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pFpInfo:

The pointer of the UCBioAPI_FASTSEARCH_FP_INFO structure that stores template information to be deleted. Refer to the UCBioAPI_FASTSEARCH_FP_INFO structure.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL
UCBioAPIERROR_FASTSEARCH_NOUSER_EXIST

■ **UCBioAPI_RemoveUserFromFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_RemoveUserFromFastSearchDB (
            [IN]  UCBioAPI_HANDLE      hHandle,
            [IN]  UCBioAPI_UINT32      nUserID);
```

Description:

All data for a specific user ID are deleted from the DB for FastSearch.
It is similar to the UCBioAPI_RemoveFpFromFastSearchDB function but there is a difference in that all data are deleted when several numbers of template data with the same user ID are stored in the DB.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

nUserID:
The user ID to be deleted.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL
UCBioAPIERROR_FASTSEARCH_NOUSER_EXIST

■ **UCBioAPI_IdentifyFIRFromFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_IdentifyFIRFromFastSearchDB (
     [IN]  UCBioAPI_HANDLE                          hHandle,
     [IN]  const UCBioAPI_INPUT_FIR_PTR             pInputFIR,
     [IN]  UCBioAPI_FIR_SECURITY_LEVEL              nSecuLevel,
     [OUT] UCBioAPI_FASTSEARCH_FP_INFO_PTR          pFpInfo,
     [IN]  UCBioAPI_FASTSEARCH_CALLBACK_INFO_PTR_0  pCallbackInfo0);
```

Description:

1:N authentication with specific FIR is attempted at the DB for FastSearch.
If authentication succeeds after calling the function, authenticated template information can be obtained. Due to the nature of 1:N authentication, authentication time may vary each time. Also, the system speed and memory may affect authentication time.
If a user wants to know the authentication process or stop authentication arbitrarily during authentication, the Callback function can be registered and used.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

pInputFIR:
The pointer of the UCBioAPI_INPUT_FIR structure that stores FIR data to be authenticated. Refer to the UCBioAPI_INPUT_FIR structure..

nSecuLevel:
Security level used during authentication is designated.
For allowed values, refer to the UCBioAPI_FIR_SECURITY_LEVEL definition.

pFpInfo:
When authentication succeeds, information of successfully authenticated template data is stored. Information such as user ID, finger ID and template number can be obtained using this value.

pCallbackInfo0:
The Callback function to be called during authentication can be designated. For more information on the Callback function, refer to the UCBioAPI_FASTSEARCH_CALLBACK_INFO_0 definition.

Returns:

UCBioAPIERROR_NONE

UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL
UCBioAPIERROR_INVALID_PARAMETER
UCBioAPIERROR_FASTSEARCH_IDENTIFY_STOP
UCBioAPIERROR_FASTSEARCH_IDENTIFY_FAIL

■ **UCBioAPI_ClearFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_ClearFastSearchDB (
            [IN]  UCBioAPI_HANDLE      hHandle);
```

Description:

The memory of the DB for FastSearch is released.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL

■ **UCBioAPI_SaveFastSearchDBToFile**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SaveFastSearchDBToFile (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [IN]  const UCBioAPI_CHAR*       szFilepath);
```

Description:

The DB for FastSearch is saved as file. If the memory DB is saved as file, loading becomes faster in the next use of the DB using the UCBioAPI_LoadFastSearchDBFromFile function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

szFilepath:

The full path of a file name to be saved as a file.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL
UCBioAPIERROR_FASTSEARCH_SAVE_DB

■ **UCBioAPI_LoadFastSearchDBToFile**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_LoadFastSearchDBToFile (
            [IN]  UCBioAPI_HANDLE            hHandle,
            [IN]  const UCBioAPI_CHAR*       szFilepath);
```

Description:

The DB for FastSearch is loaded to the memory from the file.
Only files saved using the UCBioAPI_SaveFastSearchDBToFile function can be loaded this way.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

szFilepath:
The full path of a file name to be loaded.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL
UCBioAPIERROR_FASTSEARCH_LOAD_DB

■ **UCBioAPI_GetFpCountFromFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetFpCountFromFastSearchDB (
            [IN]  UCBioAPI_HANDLE      hHandle);
            [OUT] UCBioAPI_UINT32*     pDataCount);
```

Description:

The number of templates stored in the DB for FastSearch is obtained.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pDataCount:

The pointer to store the number of templates to be obtained.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL

■ **UCBioAPI_GetFpInfoFromFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_GetFpInfoFromFastSearchDB (
            [IN]  UCBioAPI_HANDLE                   hHandle,
            [IN]  UCBioAPI_UINT32                   nDataIndex,
            [OUT] UCBioAPI_FASTSEARCH_FP_INFO_PTR   pFpInfo);
```

Description:

.Template information stored in a specific index in the DB for FastSearch is obtained.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

nDataIndex:

The index value to be obtained.

pFpInfo:

The pointer of the UCBioAPI_FASTSEARCH_FP_INFO structure to store template data information. Informations such as user ID, finger ID and template number are obtained through this value.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL
UCBioAPIERROR_INVALID_PARAMETER

■ **UCBioAPI_CheckFpExistInFastSearchDB**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_CheckFpExistInFastSearchDB (
            [IN]  UCBioAPI_HANDLE                   hHandle,
            [IN]  UCBioAPI_FASTSEARCH_FP_INFO_PTR   pFpInfo,
            [OUT] UCBioAPI_BOOL*                    pExist);
```

Description:

It examines if data with specific template information exist in the DB for FastSearch.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pFpInfo:

The pointer of the structure that stores template data to be examined.

pExist:

The pointer to get the status of existence. If it exists, the value of 1 is returned.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_INVALID_POINTER
UCBioAPIERROR_FASTSEARCH_INIT_FAIL

5.3.7. SmartCard API

APIs related to the smart card are described.
These APIs are defined in the UCBioAPI_SmartCard.h file.

- **Note – Some functions for using the smart card may not be supported depending on the firmware version of a device.**

■ UCBioAPI_SC_RFPowerOn

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_RFPowerOn (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

The RF power in the RF range is switched on within 5 seconds.
Most of smart card APIs operate with the RF power switched on.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_RFPowerOff**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_RFPowerOff (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

The RF power in the RF range is switched off within 5 seconds.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

wLed:

It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_RFFunction**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_RFFunction (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8*    pCmdBuff,
              [IN]  UCBioAPI_UINT16    nCmdLen,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, a command is transmitted and the result is returned within 5 seconds. The RF power must be switched on.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pCmdBuff:

The buffer pointer that stores commands to transmit.

nCmdLen:

The length of the buffer pointer that stores commands to transmit.

pResultBuff:

The buffer pointer to store the result value.

nResultLen:

The length of the result value.

wLed:

It designates if the result of success/failure is displayed on the LED of a device or not.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_ReadSerial**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_ReadSerial (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, the applicable serial number is obtained within 5 seconds. Since this command has the built-in function to switch the RF power on, it is not necessary to call the UCBioAPI_SC_RFPowerOn function before the function call.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pResultBuff:

The buffer pointer to store the result value.

nResultLen:

The length of the result value.

wLed:

It designates f the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_ReadBlock**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_ReadBlock (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8     BlockNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, data in the designated block are loaded within 5 seconds.
The RF power must be switched on.
This function is a Mifare card related function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

AuthMode:
It designates which key from Key A and Key B is used for authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:
The number of the sector to read. The maximum value may vary depending on the memory size of a card.

BlockNum:
The number of the block to read. It has a value ranging in 0~3.

KeyValue:
The 6 byte key value to use.

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_WriteBlock**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_WriteBlock (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8     BlockNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [IN]  UCBioAPI_UINT8*    pData,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, 16 byte data are recorded in the designated block within 5 seconds. The RF power must be switched on. This function is a Mifare card related function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

AuthMode:

It designates which key from Key A and Key B is used in authentication.

It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:

The number of the sector to write on. The maximum value may vary depending on the memory size of a card.

BlockNum:

The number of the block to read. It has a value ranging in 0 ~ 3.

KeyValue:

The 6 byte length key value to use.

pData:

The memory pointer that stores 16 byte data to write.

wLed:

It designates if the result of success/failure is displayed on the LED of a device or not.

If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_ReadSector**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_ReadSector (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, data in the designated sector are loaded within 5 seconds. The RF power must be switched on. This function is a Mifare card related function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

AuthMode:

It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:

The number of the sector to read. The maximum value may vary depending on the memory size of a card.

KeyValue:

6 byte length key value to use.

pResultBuff:

The buffer pointer to store the result value.

nResultLen:

The length of the result value.

wLed:

It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_WriteSector**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_WriteSector (
              [IN]  UCBioAPI_HANDLE     hHandle,
              [IN]  UCBioAPI_UINT8      AuthMode,
              [IN]  UCBioAPI_UINT8      SectorNum,
              [IN]  UCBioAPI_UINT8*     KeyValue,
              [IN]  UCBioAPI_UINT8*     pData,
              [IN]  UCBioAPI_UINT16     wLed);
```

Description:

If a card exists in the RF range, 48 byte length data are recorded in the designated sector within 5 seconds. The RF power must be switched on. This function is a Mifare card related function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

AuthMode:

It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:

The number of the sector to be written on. The maximum value may vary depending on the memory size of a card.

KeyValue:

6 byte length key value to use.

pData:

The memory pointer that stores 48 byte data to write.

wLed:

It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE

UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_ReadSectorFieldContent**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_ReadSectorFieldContent (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     StartSectorNum,
              [IN]  UCBioAPI_UINT8     EndSectorNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, all data inside the designated sector range are loaded within 5 seconds. The maximum allowed range is 10 sectors and the maximum data length that can be obtained is 480 bytes. (48 Bytes * 10 Sectors = 480 Bytes)
The RF power must be switched on.
This function is a Mifare card related function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

AuthMode:
It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

StartSectorNum:
The starting number of sectors to read. The maximum value may vary depending on the memory size of a card.

EndSectorNum:
The ending number of sectors to read. The maximum value may vary depending on the memory size of a card.

KeyValue:
6 byte length key value to use.

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designates of the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_WriteSectorFieldContent**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_WriteSectorFieldContent (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     StartSectorNum,
              [IN]  UCBioAPI_UINT8     EndSectorNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [IN]  UCBioAPI_UINT8*    pData,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, values are written on all data inside the designated sector range. The maximum allowed range is 10 sectors and the maximum data length that can be obtained is 480 bytes. (48 Bytes * 10 Sectors = 480 Bytes)
The RF power must be switched on.
This function is a Mifare card related function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

AuthMode:
It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

StartSectorNum:
The starting number of sectors to be written on. The maximum value may vary depending on the memory size of a card.

EndSectorNum:
The ending number of sectors to be written on. The maximum value may vary depending on the memory size of a card.

KeyValue:
The 6 byte length key value to use.

pData:
The memory pointer that stores 48 bytes data to write.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_PreValue**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_PreValue (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8     BlockNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [IN]  UCBioAPI_UINT8*    pData,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, the designated block is switched to the Value mode and 4 byte pData value is written within 5 seconds.

If the pData value to be written is 0x00000000, a value as in next is written on the applicable block. (0x00000000FFFFFFFF00000000FFFFFFFF00FF00FF)

The block switched to the Value mode this way can use Value mode functions such as UCBioAPI_SC_ReadValue and UCBioAPI_SC_IncrementValue. For more detailed information on the Value mode, refer to Using Smart Card in this document.

The RF power must be switched on.

This function is a Mifare card related function.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

AuthMode:

It designates which key from Key A and Key B is used in authentication.

It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:

The number of the sector to be written on. The maximum value may vary depending on the memory size of a device.

BlockNum:

The number of the block to be written on. It has a value ranging in 0 ~ 3.

KeyValue:

6 byte length key value to use.

pData:
The memory pointer that stores 4 byte data to write.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_ReadValue**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_ReadValue (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8     BlockNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, 4 byte value data in the designated block are loaded.
The RF power must be switched on and the designated block must be in the Value mode.
This function is a Mifare card related function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

AuthMode:
It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:
The number of the sector to read. The maximum value may vary depending on the memory size of a card.

BlockNum:
The number of the block to read. It has a value ranging in 0 ~ 3.

KeyValue:
6 byte length key value to use.

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_IncrementValue**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_IncrementValue (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8     BlockNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [IN]  UCBioAPI_UINT8*    pData,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, 4 byte data in the designated block are increased by the designated value within 5 seconds.
The RF power must be switched on and the designated block must be in the Value mode.
This function is a Mifare card related function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

AuthMode:
It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:
The number of the sector to be written on. The maximum value may vary depending on the memory size of a card.

BlockNum:
The number of the block to be written on. It has a value ranging in 0 ~ 3.

KeyValue:
6 byte length key value to use.

pData:
The memory pointer that stores 4 byte value data to be increased.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.

If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_DecrementValue**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_DecrementValue (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8     BlockNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [IN]  UCBioAPI_UINT8*    pData,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, 4 byte data in the designated block is decreased by the designated value.
The RF power must be switched on and the designated block must be in the Value mode.
This function is a Mifare card related function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

AuthMode:
It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:
The number of the sector to be written on. The maximum value may vary depending on the memory size of a device.

BlockNum:
The number of the block to be written on. It has a value ranging in 0 ~ 3.

KeyValue:
6 byte length key value to use.

pData:
The memory pointer that stores 4 byte value data to be decreased.

wLed:
It designates if the result of failure/success is displayed on the LED of a device or not.

If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_WriteSectorTrailer**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_WriteSectorTrailer (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     AuthMode,
              [IN]  UCBioAPI_UINT8     SectorNum,
              [IN]  UCBioAPI_UINT8*    KeyValue,
              [IN]  UCBioAPI_UINT8*    NewAccessBit,
              [IN]  UCBioAPI_UINT8*    NewKeyA,
              [IN]  UCBioAPI_UINT8*    NewKeyB,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, the sector trailer area in the designated sector is modified within 5 seconds.
The sector trailer includes key A, access bits and key B of an applicable sector.
Success/Failure is determined according to access rights of an applicable sector.
The RF power must be switched on.
This function is a Mifare card related function.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

AuthMode:
It designates which key from Key A and Key B is used in authentication.
It can have the value of either UCBioAPI_SC_USE_KEY_A(0x60) or UCBioAPI_SC_USE_KEY_B(0x61).

SectorNum:
The number of the sector to be written on. The maximum value may vary depending on the memory size of a card.

KeyValue:
6 byte length key value to use.

NewAccessBit:
4 byte length buffer pointer that stores new access bits.

NewKeyA:
6 byte length buffer pointer that stores new key A.

NewKeyB:
6 byte length buffer pointer that stores new key B.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_ReqA**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_ReqA (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a type A card exists in the RF range, ATQ is obtained from the card within 5 seconds.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pResultBuff:

The buffer pointer to store the result value.

nResultLen:

The length of the result value.

wLed:

It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_WupA**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_WupA (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a type A card exists in the RF range, ATQ is obtained from the card within 5 seconds.
It is identical to the UCBioAPI_SC_ReqA function but there is a difference in that a card in halt state responds.
The Rd power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designated if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_Select**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_Select (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

With ATQ obtained from a type A card, if a type A card exists in the RF range, the SAK value and UID value of the card are obtained from the card within 5 seconds.
The length of SAK is 1 byte and the length of UID is variable.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

pResultBuff:
The buffer pointer to store the result value.
SAK(1byte) + UID are stored sequentially.

nResultLen:
The length of the result value.

wLed:
It designates of the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_HaltA**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_HaltA (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

The type A card in the select state is switched to the halt state.
The card that became halt state this way does not respond to the UCBioAPI_SC_ReqA function but it responds only to the UCBioAPI_SC_WupA function.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_Rats**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_Rats (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     fsdi,
              [IN]  UCBioAPI_UINT8     cid,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a card exists in the RF range, the ATS (Answer To Select) value of the card is obtained from the card within 5 seconds. ATS can be obtained when the SAK value is 0x2X (ISO14443-4 supported).
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

Fsdi:
The maximum size of the frame that PCS can receive can be determined through the setting of FSDI (Frame Size for proximity coupling Device Integer). (Value ranging in 0x00 ~ 0x0F)

cid:
It is possible to call an individual card selectively through CID (Card Identifier). (Value ranging in 0x00 ~ 0x0F)

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_PpsRequest**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_PpsRequest (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     cid,
              [IN]  UCBioAPI_UINT8     pps0,
              [IN]  UCBioAPI_UINT8     pps1,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a type A card exists in the RF range, the PPSS value of the card is obtained from the card within 5 seconds. If a parameter that can be changed through the ATS value, it can be used by PCD. That is, if a higher Baud rate is supported at DS and DR which are selected parameters in ATS, the Baud rate for both directions can be increased independently by 2, 4, 8 times.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

cid:
The CID (Card Identifier) value selected during rats call. (Values ranging in 0x00 ~ 0x0F)

pps0:
It represents if PPS1 is transmitted or not.
It represents transmission if it is 0x11. It represents no transmission if it is 0x01.

pps1:
Upper 2 bytes mean DRI and lower 2 bytes mean DS.

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:

It designates of the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_BlockFormat**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_BlockFormat (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     pcb,
              [IN]  UCBioAPI_UINT8     CidOrNad,
              [IN]  UCBioAPI_UINT8*    inf,
              [IN]  UCBioAPI_UINT8     infLen,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a type A card exists in the RF range, data are loaded from the card within 5 seconds.
Command related to block exchange in the data layer (T=1).
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:

The Handle value of UCBioAPI SDK.

pcb:

It means PCB (Protocol Control Byte). (Required)
Information required in controlling data transmission is available. It is divided into I – Block, R – Block, S – Block. I – Block is used to read data and PCB value is transmitted by toggling Bit0 as in 0x0A, 0x0B, 0x0A.

CidOrNad:

CID or NAD value designated at PICC with Rats. (Optional)

inf:

It means INF (Information Field).
To read an applicable file, a special format must be known.

infLen:

The length of inf.

pResultBuff:

The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designates of the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:
UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_Deselect**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_Deselect (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT8     CidOrNad,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a type A card exists in the RF range, the card is deactivated within 5 seconds.
If an active state card is deselected, the card does not respond to BlockFormat as long as it is within the RF range. The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

CidOrNad:
CID or NAD value designated at PICC with Rats.

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

■ **UCBioAPI_SC_TypeA_ActiveState**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_TypeA_ActiveState (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [OUT] UCBioAPI_UINT8*    pResultBuff,
              [OUT] UCBioAPI_UINT16*   nResultLen,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:

If a type A card exists in the RF range, ATS is obtained by implementing ReqA, Select and Rats simultaneously within 5 seconds.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

hHandle:
The Handle value of UCBioAPI SDK.

pResultBuff:
The buffer pointer to store the result value.

nResultLen:
The length of the result value.

wLed:
It designates if the result of success/failure is displayed on the LED of a device or not.
If UCBioAPI_SC_LED_TOGGLE(1) is set, the LED of a device changes to blue upon success.

Returns:

UCBioAPIERROR_NONE
UCBioAPIERROR_INVALID_HANDLE
UCBioAPIERROR_FUNCTION_NOT_SUPPORTED
UCBioAPIERROR_DEVICE_NOT_OPENED
UCBioAPIERROR_SC_FUNCTION_FAILED
UCBioAPIERROR_SC_NOT_SUPPORTED_DEVICE
UCBioAPIERROR_SC_NOT_SUPPORTED_FIRMWARE

6.API Reference for COM

This chapter describes properties and methods to use a COM module, UCBioBSPCOM.dll.

6.1. UCBioBSP Object

As the main object to use UCBioBSPCOM.dll, it provides the version information and basic setting of UCBioBSP SDK and operates as the default object to get various function-by-function child objects. Therefore, to use SDK, this object must be declared.

6.1.1. Methods

Various methods of UCBioBSP Object are described.

■ **SetSkinResource**

Prototype:

```
HRESULT SetSkinResource(BSTR bszSkinPath);
```

Description:

A skin for the user interface of UCBioAPI SDK is changed.
UCBioAPI SDK uses a skin type UI for the screen used in registration and authentication. Therefore, to use UI in other languages and forms instead of the UI provided by UCBioBSP SDK, a user defined skin can be created and used.

Parameters:

bszSkinPath:
The full path of the skin file to be changed.

Related properties:

ErrorCode, ErrorDescription

■ **SetCaptureQuality**

Prototype:

```
HRESULT SetCaptureQuality(
                          long nScanQuality,
                          long nEnrollQuality,
                          long nVerifyQuality,
                          long nIdentifyQuality);
```

Description:

You can set input fingerprint quality values on SDK. Set value ranges from 1 to 100, and

value 0(zero) is ignored.(Maintained old value)

Parameters:
nScanQuality : minimum quality value of input finger print image quality.
nEnrollQuality : minimum minutiae quality fo input fingerprint for enroll
nVerifyQuality : minimum minutiae quality fo input fingerprint for 1:1 matching
nIdentifyQuality : minimum minutiae quality fo input fingerprint for 1:N matching

*** now support only nScanQuality, nEnrollQuality.**

Returns: UCBioAPIERROR_TRUE

6.1.2. Properties

Various properties of UCBioBSP Object are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:
Values for errors occurred during the setting of executed methods and properties are stored.
The value of 0 means success and all other values mean failure.
Errors occurred during the setting of the method and property of a child object can also be obtained using this value.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:
Values for errors occurred during the setting of executed methods and properties are stored as character string.
It is used to output error values on ErrorCode in character string.
As in ErrorCode, errors occurred during the setting of the method and property of a child object can also be obtained using this value.

■ **Device**

Prototype:

```
[ReadOnly]  VARIANT       Device;
```

Description:

The interface of the object that has a collection of device relate commands is obtained. This interface is obtained to perform functions such as start and end of device use, device selection and option value setting. For more detailed information, refer to the IDevice description.

■ **Extraction**

Prototype:

```
[ReadOnly]  VARIANT       Extraction;
```

Description:

The interface related to a function to extract special features after obtaining a fingerprint is obtained. To perform a function related to fingerprint acquisition and registration, this interface is obtained and used. For more detailed information, refer to the IExtraction description.

■ **Matching**

Prototype:

```
[ReadOnly]  VARIANT       Matching;
```

Description:

The interface related to a function to perform authentication using fingerprint data is obtained. To use perform a function related to fingerprint authentication, this interface is obtained and used. For more detailed information, refer to the IMatching description.

■ **FPData**

Prototype:

```
[ReadOnly]  VARIANT       FPData;
```

Description:

The interface related to a function to obtain or convert fingerprint data is obtained. To convert fingerprint data into a different type, this function is obtained and used. For more detailed information, refer to the IFPData description.

■ **FPImage**

Prototype:

```
[ReadOnly]  VARIANT       FPImage;
```

Description:

The interface related to a function to obtain fingerprint data as image or store them as image file is obtained. To convert fingerprint data into image, this interface is obtained and used. For more detailed information, refer to the IFPImage description.

■ **FastSearch**

Prototype:

```
[ReadOnly]  VARIANT       FastSearch;
```

Description:

The interface related to the FastSearch function to perform 1:N high-speed authentication is obtained. To use a function related to FastSearch related DB management and authentication implementation, this interface is obtained and used. For more detailed information, refer to the IFastSearch description.

■ **SmartCard**

Prototype:

```
[ReadOnly]  VARIANT       SmartCard;
```

Description:

The interface related to the smart card is obtained. To use a function to store data at the smart card and load data, this function is be obtained and used. For more detailed information, refer to the ISmartCard description.

■ **CheckValidityModule**

Prototype:

```
[ReadOnly]  BOOL          CheckValidityModule;
```

Description:

The validity of UCBioBSP SDK is examined and that value is kept.

If this value is TRUE, there is no problem with validity. But if this value is FALSE, there is a possibility that the UCBioBSP.dll file, a core module of SDK, may have been changed or modified and it requires verification.

■ **MajorVersion**

Prototype:

```
[ReadOnly]  BSTR          MajorVersion;
```

Description:

The major version numbers of UCBioBSP SDK is stored as string. For example, if the version of SDK is v3.4.0.2, "3.4" is placed in the major version location and "0.2" is placed in the minor version location.

■ **MinorVersion**

Prototype:

```
[ReadOnly]  BSTR          MinorVersion;
```

Description:

The minor version numbers of UCBioBSP SDK is stored as string. For example, if the version of SDK is v3.4.0.2, "3.4" is placed in the major version location and "0.2" is placed in the minor version location.

■ **BuildNumber**

Prototype:

```
[ReadOnly]  BSTR          BuildNumber;
```

Description: not support.

■ **MaxFingersForEnroll**

Prototype:

```
[Read/Write] long         MaxFingersForEnroll;
```

Description:

The maximum number of fingers allowed for registration during fingerprint registration is designated.

For example, if this value is set as 2, only up to 2 fingers can be registered when registering fingerprints using the Enroll method of IExtraction.

The default value is 10.

Related methods:

IExtraction.Enroll

■ **NecessaryEnrollNum**

Prototype:

```
[Read/Write] long        NecessaryEnrollNum;
```

Description:

The minimum number of fingers required for registration during fingerprint registration is designated.
This value must be less than or equal to the MaxFingersForEnroll value.
For example, if this value is set as 2, at least 2 fingers need to be registered to complete a registration process when registering fingerprints using the Enroll method of IExtraction.
The default value is 2.

Related methods:

IExtraction.Enroll

■ **SamplesPerFinger**

Prototype:

```
[Read/Write] long        SamplesPerFinger;
```

Description:

The number of samples stored per finger during fingerprint registration is designated. Currently, it is fixed at 2 and modification is not allowed.

■ **DefaultTimeout**

Prototype:

```
[Read/Write] long        DefaultTimeout;
```

Description:

The default maximum time for which a device operates to acquire a fingerprint during fingerprint authentication and registration is designated in ms unit. Timeout can be set separately later during a function call but this value is used if -1 is set now.
The default value is 10000 (10 seconds).

Related methods:

IExtraction.Enroll, IExtraction.Capture, IMatching.Verify

■ **SecurityLevelForEnroll / SecurityLevelForVerify / SecurityLevelForIdentify**

Prototype:

```
[Read/Write] long        SecurityLevelForEnroll;
[Read/Write] long        SecurityLevelForVerify;
[Read/Write] long        SecurityLevelForIdentify;
```

Description:

An authentication security level to be used in fingerprint registration / authentication / 1:N authentication can be set for each one of them. Allowed values are shown below.

```
1 -        LOWEST
2 -        LOWER
3 -        LOW
4 -        BELOW_NORMAL
5 -        NORMAL
6 -        ABOVE_NORMAL
7 -        HIGH
8 -        HIGHER
9 -        HIGHEST
```

As a default value, Enroll/Verify has 5 and Identify has 6.

Related methods:

IExtraction.Enroll, IMatching.Verify, IMatching.VerifyMatch

■ **WindowStyle**

Prototype:

```
[Read/Write] long        WindowStyle;
```

Description:

The type of displaying a Window on the screen can be designated. It determines if a Window is launched as pop-up type or only a fingerprint is displayed in an area of another Window.

```
0 -        POPUP
1 -        INVISIBLE
```

If POPUP is designated, a new Window is launched generally to launch fingerprint registration and authentication. But if INVISIBLE can be designated for the Capture method of IExtraction and Verify method of IMatching, then fingerprint input or authentication can be performed while not displaying UI on the screen. Also, if INVISIBLE is designated like

this, fingerprint image is allowed to be displayed on that Window assuming that the FingerWnd property value is not NULL.

Related methods:
IExtraction.Enroll, IExtraction.Capture, IMatching.Verify, IDevice.Adjust

■ **WindowOption**

Prototype:

```
[Read/Write] BOOL        WindowOption(long Option);
```

Description:
The flag for the type of displaying a Window on the screen can be designated. Settings such as not allowing a fingerprint to be displayed or removing the Welcome page during fingerprint registration can be made. These three flags can be designated repeatedly. A value shown in below is designated as the option value, and the TRUE/FALSE value is designated or the value is obtained.

```
0x00010000 – NO_FPIMG
0x00020000 – NO_WELCOME
0x00040000 – NO_TOPMOST
```

0x00010000 - NO_FPIMG:
When fingerprint image is preferred not to be displayed on the screen for security reasons, this style is designated.
0x00020000 - NO_WELCOME:
The first page "Welcome page" from fingerprint registration UIs is not displayed.
0x00040000 - NO_TOPMOST:
All UIs are currently to be opened on top of others. To open it as a general Window type, this style is designated.

Related methods:
IExtraction.Enroll, IExtraction.Capture, IMatching.Verify, IDevice.Adjust

■ **ParentWnd**

Prototype:

```
[Read/Write] long        ParentWnd;
```

Description:
When a Window for fingerprint registration and authentication is launched, the Handle of

the parent Window that is the base of that Window is designated. The default value is NULL.

Related methods:
IExtraction.Enroll, IExtraction.Capture, IMatching.Verify, IDevice.Adjust

■ **FingerWnd**

Prototype:

```
[Read/Write] long        FingerWnd;
```

Description:
If WindowStyle is set as INVISIBLE(1), the Handle of a specific Window to draw fingerprint image is designated. The default value is NULL.

Related methods:
IExtraction.Capture, IMatching.Verify

■ **CaptionMsg**

Prototype:

```
[Read/Write] BSTR        CaptionMsg;
```

Description:
When a user selects CANCEL during fingerprint registration, a message to be displayed on the caption of a Window to display the cancel message is designated.

Related methods:
IExtraction.Enroll

■ **CancelMsg**

Prototype:

```
[Read/Write] BSTR        CancelMsg;
```

Description:
When a user selects CANCEL during fingerprint registration, a cancel message to be displayed is designated.

Related methods:
IExtraction.Enroll

■ **FPForeColor / FPBackColor**

Prototype:

```
[Read/Write] BSTR        FPForeColor;
[Read/Write] BSTR        FPBackColor;
```

Description:

The fingerprint color and background value of fingerprint image to be displayed on the screen are designated as RGB values in character string.
If it is designated as "FFFFFF", each 2 byte can be considered to be an RGB value in HEX.

Related methods:

IExtraction.Enroll, IExtraction.Capture, IMatching.Verify

■ **DisableFingerForEnroll**

Prototype:

```
[Read/Write] BOOL        DisableFingerForEnroll(long nFingerID);
```

Description:

Fingers to be prohibited from registration during fingerprint registration can be designated. It is the array value that can store 10 fingers in total, and the registration status for each finger is designated by 0 or 1.
It increases from index number 1 in the order of thumb, index, middle, ring and little of the right hand and from index number 6 in the order of thumb, index, middle, ring and little of the left hand.
For example, to prohibit the left hand thumb from registration, designate it as shown below.

```
objUCBioBSP.DisableFingerForEnroll(6) = True;
```

But, in the fingerprint modification mode, registration status for an already registered finger is displayed even if it is prohibited from registration.

Related methods:

IExtraction.Enroll

6.2. IDevice Interface

The interface of an object with a collection of device related commands. To perform functions such as starting and ending of device use, device selection and option value setting, this interface is obtained and used.

6.2.1. Methods

Various methods of IDevice interface are described.

■ **Open**

Prototype:

```
HRESULT Open(long nDeviceID);
```

Description:

A desired device is opened and initialized.

Parameters:

nDeviceID:

The ID of a device to be opened. Refer to UCBioAPI_DEVICE_ID.

Related properties:

ErrorCode, ErrorDescription

OpenedDeviceID, ImageWidth, ImageHeight, Brightness, Contrast, Gain

■ **Close**

Prototype:

```
HRESULT Close(long nDeviceID);
```

Description:

An opened device is closed and its use is terminated.

Parameters:

nDeviceID:

The ID of a device to be closed. Refer to UCBioAPI_DEVICE_ID.

Related properties:

ErrorCode, ErrorDescription

■ **Enumerate**

Prototype:

```
HRESULT Enumerate();
```

Description:

A list is created by searching all devices installed in the current system. The created list can be obtained through a related property.

Related properties:

ErrorCode, ErrorDescription
EnumCount, EnumDeviceID, EnumDeviceNameID, EnumDeviceInstance, EnumDeviceName, EnumDeviceDescription, EnumDeviceDll, EnumDeviceSys, EnumDeviceAutoOn, EnumDeviceBrightness, EnumDeviceContrast, EnumDeviceGain

■ **Adjust**

Prototype:

```
HRESULT Adjust();
```

Description:

A UI that adjusts the brightness of a currently opened device is launched. Since currently supported devices internally undergo the automatic brightness adjustment procedure, this function can not be used.

Related properties:

ErrorCode, ErrorDescription

6.2.2. Properties

Various properties of IDevice interface are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 means success and all other values mean failure.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:

Values on errors occurred during the setting of executed methods and properties are stored as character string.

It is used to output error values on ErrorCode as character string.

■ **EnumCount**

Prototype:

```
[ReadOnly]  long          EnumCount;
```

Description:

The number of devices searched after calling the IDevice.Enumerate method is stored.

It is used only after the IDevice.Enumerate method is called.

Related methods:

IDevice.Enumerate

■ **EnumDeviceID**

Prototype:

```
[ReadOnly]  long          EnumDeviceID(long nIndex);
```

Description:

The ID list of devices searched after calling the IDevice.Enumerate method is stored.

nIndex can have a value ranging in 0 ~ (EnumCount -1).
It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **EnumDeviceNameID**

Prototype:

```
[ReadOnly]  long          EnumDeviceNameID(long nIndex);
```

Description:
The name ID list of devices searched after calling the IDevice.Enumerate method is stored.
nIndex can have a value ranging in 0 ~ (EnumCount -1).
It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **EnumInstance**

Prototype:

```
[ReadOnly]  long          EnumDeviceInstance(long nIndex);
```

Description:
The instance list of devices searched after calling the IDevice.Enumerate method is stored.
Since device-by-device instance is not currently supported, it is not used.
It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **EnumDeviceName**

Prototype:

```
[ReadOnly]  BSTR          EnumDeviceName(long nIndex);
```

Description:
The name list of devices searched after calling the IDevice.Enumerate method is stored.
nIndex can have a value ranging in 0 ~ (EnumCount -1).
It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **EnumDeviceDescription**

Prototype:

```
[ReadOnly]  BSTR        EnumDeviceDescription(long nIndex);
```

Description:
The description list of devices searched after calling IDevice.Enumerate method is stored.
nIndex can have a value ranging in 0 ~ (EnumCount -1).
It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **EnumDeviceDll / EnumDeviceSys**

Prototype:

```
[ReadOnly]  BSTR        EnumDeviceDll(long nIndex);
[ReadOnly]  BSTR        EnumDeviceSys(long nIndex);
```

Description:
The DLL name list and Sys name list of devices searched after calling the IDevice.Enumerate method is stored.
nIndex can have a value ranging in 0 ~ (EnumCount -1).
It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **EnumDeviceAutoOn**

Prototype:

```
[ReadOnly]  long        EnumDeviceAutoOn(long nIndex);
```

Description:
The list of values to determine if devices searched after calling the IDevice.Enumerate method support AutoOn is stored. Currently, values for AutoOn are not supported.
nIndex can have a value ranging in 0 ~ (EnumCount -1).

It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **EnumDeviceBrightness / EnumDeviceContrast / EnumDeviceGain**

Prototype:

```
[ReadOnly]  long        EnumDeviceBrightness(long nIndex);
[ReadOnly]  long        EnumDeviceContrast(long nIndex);
[ReadOnly]  long        EnumDeviceGain(long nIndex);
```

Description:
The brightness / contrast / gain value list of devices searched after calling the IDevice.Enumerate method is stored.
nIndex can have a value ranging in 0 ~ (EnumCount -1).
It is used only after the IDevice.Enumerate method is called.

Related methods:
IDevice.Enumerate

■ **OpenedDeviceID**

Prototype:

```
[ReadOnly]  long        OpendedDeviceID;
```

Description:
The ID of a currently opened device is stored.

■ **DeviceNameID / DeviceInstance**

Prototype:

```
[ReadOnly]  long        DeviceNameID(long nDeviceID);
[ReadOnly]  long        DeviceInstance(long nDeviceID);
```

Description:
The device name ID and device instance are obtained from a given nDeviceID.
Each of values represents the lower 2 bytes and upper 2 bytes of nDeviceID.
Since device-by-device instance is not currently supported, it is not used.

■ **DeviceID**

Prototype:

```
[ReadOnly]  long          DeviceID(long nNameID, long nInstance);
```

Description:

The device ID is obtained using given nNameID and nInstance.
Since device-by-device instance is not currently supported, it is not used.

■ **ImageWidth / ImageHeight**

Prototype:

```
[ReadOnly]  long          ImageWidth(long nDeviceID);
[ReadOnly]  long          ImageHeight(long nDeviceID);
```

Description:

The image size of the device is obtained using the given nDeviceID.
nDeviceID must be identical to a currently opened device ID and it must be used while a device is opened.

Related methods:

IDevice.Open, IDevice.OpenedDeviceID

■ **Brightness / Contrast / Gain**

Prototype:

```
[Read/Write] long         Brightness(long nDeviceID);
[Read/Write] long         Contrast(long nDeviceID);
[Read/Write] long         Gain(long nDeviceID);
```

Description:

The brightness / contrast / gain value of the device is obtained or designated using the given nDeviceID.
nDeviceID must be identical to a currently opened device ID and it must be used while a device is opened.

Related methods:

IDevice.Open, IDevice.OpenedDeviceID

■ **IsFingerExisted**

Prototype:

```
[ReadOnly]  BOOL          IsFingerExisted;
```

Description:
If a finger is placed on a currently opened device is examined and the examination result is stored.
An examination starts at the time of this property call.
It must be used while a device is opened.

Related methods:
IDevice.Open

6.3. IExtraction Interface

The interface related to a function that extracts special features by obtaining fingerprint image. To perform a function related to fingerprint acquisition and registration, this interface is obtained and used.

6.3.1. Methods

Various methods of the IExtraction interface are described.

■ **Auto Detect**

Prototype:

```
HRESULT SetAutoDetect(long nFlag);
```

Description:

Setting for Auto detection for Finger

Parameters:

nFlag:

Flag for using auto detection

```
0 -        Not Use
1 -        Use Autodetect
```

Related properties:

Capture, Enroll

FIR, FIRLength, TextFIR

Related methods:

IDevice.Open

■ **Capture**

Prototype:

```
HRESULT Capture([in, optional]long nPurpose);
```

Description:

FIR Handle is created by acquiring one fingerprint from a currently opened device. Data acquired this way can be obtained using FIR or TextFIR property. Since this method uses a device, a device must be opened before use.

Parameters:

nPurpose:

The purpose value of FIR data can be designated. Allowed values are shown below.

```
1 -       VERIFY
2 -       IDENTIFY
3 -       ENROLL
4 -       ENROLL_FOR_VERIFICATION_ONLY
5 -       ENROOL_FOR_IDENTIFICATION_ONLY
6 -       AUDIT
16 -      UPDATE
```

Each of values is used only for reference on FIR data and it does not have any effect on authentication. If this value is designated as Enroll related purpose, it operates in the same way as the Enroll method.
For more detailed description, refer to the UCBioAPI_FIR_PURPOSE definition.

Related properties:
ErrorCode, ErrorDescription
FIR, FIRLength, TextFIR

Related methods:
IDevice.Open

■ **Enroll**

Prototype:

```
HRESULT Enroll(VARIANT payload, [in, optional]VARIANT storedFIR);
```

Description:
FIR data are created by acquiring fingerprints of one person from a currently opened device. Since fingerprints are from one person, fingerprints for several fingers are registered at the same time and one FIR data is created from them. Fingerprints registered this way can be obtained using FIR or TextFIR property. Since this method uses a device, a device must be opened before use.

Parameters:
payload:
Payload data to be stored at FIR data to be created. They can be either character string or binary data.
This value can have NULL. If NULL is designated, the Payload value of existing data is kept.

storedFIR:
To modify existing FIR data, this value is designated. If this value is not designated, a new

fingerprint is entered for input.

Related properties:
ErrorCode, ErrorDescription
FIR, FIRLength, TextFIR

Related methods:
IDevice.Open

6.3.2. Properties

Various properties of the IExtraction interface are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 represents success and all other values represent failure.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored as character string.

It is used to output error values for ErrorCode as character string.

■ **FIR**

Prototype:

```
[ReadOnly]  VARIANT       FIR;
```

Description:

FIR data acquired after calling the Capture or Enroll method are obtained as binary stream data.

The length of data are stored at the FIRLength property.

FIR format to obtain must be designated in advance at the FIRFormat property.

Related methods:

Capture, Enroll

Related properties:

FIRLength, FIRFormat

■ **FIRLength**

Prototype:

```
[ReadOnly]  long          FIRLength;
```

Description:

The length of binary stream data acquired after calling the Capture or Enroll method is obtained.

Related methods:

Capture, Enroll

Related properties:

FIR

■ **TextFIR**

Prototype:

```
[ReadOnly]  BSTR          TextFIR;
```

Description:

FIR data acquired after calling the Capture or Enroll method are obtained as character string in text string type. Since it is a character string, its length is not separately designated.

FIR format to obtain must be designated in advance at the FIRFormat property.

Related methods:

Capture, Enroll

Related properties:

FIRFormat

■ **FIRFormat**

Prototype:

```
[Read/Write] long         FIRFormat;
```

Description:

The data format of FIR to obtain is designated. When the structure of FIR data changes later, this format value changes and the lower compatibility can be maintained. Only the STANDARD(1) format is currently supported.

Related properties:

FIR, TextFIR

6.4. IMatching Interface

The interface related to a function that performs authentication using fingerprint data. To perform a function relates to fingerprint authentication, this interface is obtained and used.

6.4.1. Methods

Various methods of the IMatching interface are described.

■ **VerifyMatch**

Prototype:

```
HRESULT VerifyMatch (VARIANT processedFIR, VARIANT storedFIR);
```

Description:

Two previously acquired FIR data are compared each other and the authentication result is obtained. Upon successful authentication, the Payload value stored at FIR data for registration can also be obtained.

Parameters:

processedFIR:

FIR data requiring authentication. They can be either FIR data in binary stream type or TextFIR data in text string type. Either one of them can be used without making a difference.

storedFIR:

FIR data for registration requiring authentication. They can be either FIR data in binary stream type or TextFIR data in text string type. Either one of them can be used without making a difference.

Related properties:

ErrorCode, ErrorDescription
MatchingResult, IsPayloadExisted, Payload, PayloadLength, TextPayload

■ **Verify**

Prototype:

```
HRESULT Verify (VARIANT storedFIR);
```

Description:

Previously acquired FID data and fingerprint data entered in real time from the current device are compared and the authentication result is obtained. Upon successful authentication, the Payload value stored at FIR data for registration can also be obtained. It

is nearly identical to the VerifyMatch method but there is a difference in that authentication is implemented by directly accepting a fingerprint in real time.
That is, it can be considered that the IExtraction.Capture method and the IMathing.VerifyMatch method are internally implemented at the same time. Since this function uses a device, a device must be opened before use.

Parameters:
storedFIR:
FIR data for registration requiring authentication. They can be either FIR data in binary stream type or TextFIR data in text string type. Either one of them can be used without making a difference.

Related properties:
ErrorCode, ErrorDescription
MatchingResult, IsPayloadExisted, Payload, PayloadLength, TextPayload

Related methods:
IDevice.Open

6.4.2. Properties

Various properties of the IMatching interface are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 represents success and all other values represent failure.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored as character string.

It is used to output error values for ErrorCode as character string.

■ **MatchingResult**

Prototype:

```
[ReadOnly]  BOOL          MatchingResult;
```

Description:

After the Verify or VerifyMatch method is called, the result for that authentication is stored.

Related methods:

Verify, VerifyMatch

■ **IsPayloadExisted**

Prototype:

```
[ReadOnly]  BOOL          IsPayloadExisted;
```

Description:

If authentication succeeds after calling the Verify or VerifyMatch method, Payload stored inside FIR can be obtained and the value to determine if the Payload value exists is stored.

Related methods:
Verify, VerifyMatch

Related properties:
Payload, PayloadLength, TextPayload

■ **Payload**

Prototype:

```
[ReadOnly]  VARIANT       Payload;
```

Description:
If authentication succeeds after calling the Verify or VerifyMatch method, Payload stored inside FIR can be obtained and that Payload value is stored as binary stream type.
The length of binary stream data is stored in the PayloadLength property.

Related methods:
Verify, VerifyMatch

Related properties:
PayloadLength

■ **PayloadLength**

Prototype:

```
[ReadOnly]  long          PayloadLength;
```

Description:
If authentication succeeds after calling the Verify or VerifyMatch method, Payload stored inside FIR can be obtained and the length for Payload data in binary stream type is stored.

Related methods:
Verify, VerifyMatch

Related properties:
Payload

■ **TextPayload**

Prototype:

```
[ReadOnly]  BSTR          TextPayload;
```

Description:
If authentication succeeds after calling the Verify or VerifyMatch method, Payload stored inside FIR can be obtained and the Payload value is stored as character string in text string type.

Related methods:
Verify, VerifyMatch

6.5. IFPData Interface

The interface related to a function to obtain or convert fingerprint data. To convert fingerprint data into a different type, this interface is obtained and used.

6.5.1. Methods

Various methods of the IFPData interface are described.

■ **ClearFPData**

Prototype:

```
HRESULT ClearFPData();
```

Description:

Initialize all properties in FPData

Parameters:

없음

Related properties:

All

■ **Export**

Prototype:

```
HRESULT Export (VARIANT storedFIR, nDestFPDataType);
```

Description:

Desired type template information is obtained from FIR data. After calling the method, each of template informations can be obtained using various properties.

Since FIR data generally hide internal data through encryption, template-by-template information is not known. Therefore, to process data template-by-template as in other application programs ad in the terminal, data need to be converted using this method before use.

Parameters:

storedFIR:

FIR data requiring conversion. They can be either FIR data in binary stream type or TextFIR data in text string type. Either one of them can be used without making a difference.

nDestFPDataType:

The type of template data to convert is designated.

For allowed values, refer to the UCBioAPI_TEMPLATE_TYPE definition.

Related properties:
ErrorCode, ErrorDescription
TotalFingerCount, SampleNumber, FingerID, FPSampleData, FPSampleDataLength

■ **Import**

Prototype:

```
HRESULT Import (
            BOOL bInitialize,
            long nFingerID,
            long nPurpose,
            long nSrcFPDataType,
            long nFPDataSize,
            VARIANT FPData1,
            [in, optional] VARIANT FPData2);
```

Description:

FIR data are created using one template information or several numbers of template informations in a specific type.

After calling the method, FIR with template information can be obtained through various properties.

In general, UCBioBSP SDK does not allow direct authentication using template and all data are required to be converted to FIR type before use. Therefore, it is necessary to convert data into FIR data using this method.

Data created this way can be obtained using FIR or TextFIR property.

Parameters:

bInitialize

If FIR data are initialized and created or not is designated.

If this value is False, template data added now continue to be added to internally created FIR data and one FIR data with several numbers of template data is created. However, if this value is True, all existing FIRs are deleted and new FIR is created.

nFingerID:

The finger ID information of the template to be added. For related values, refer to UCBioAPI_FINGER_ID.

nPurpose:

The purpose value of FIR data to convert can be designated.

These values are used only for reference on FIR data and they do not have any effect on authentication. For more detailed description, refer to the UCBioAPI_FIR_PURPOSE definition.

nSrcFPDataType:

The type information of the template to be added. For related values, refer to UCBioAPI_TEMPLATE_TYPE.

FPData1:
Template data to be added. (Binary stream data)

FPData2:
The second template data of a finger to be added. (Binary stream data)
It is not necessary to designate this value as option. In such a case, the value of SamplesPerFinger becomes 1 internally for FIR.

Related properties:
ErrorCode, ErrorDescription
FIR, FIRLength, TextFIR

■ **CreateTemplate**

Prototype:

```
HRESULT CreateTemplate (
            VARIANT capturedFIR,
            VARIANT storedFIR,
            [in, optional] VARIANT payload);
```

Description:

A new FIR Handle is created by merging new FIR data with existing FIR data or replacing with new FIR data. Also, it is used when existing FIR is replaced with new Payload. Data created this way can be obtained using FIR or TextFIR property.

Parameters:

capturedFIR:

FIR data to newly replace. They can be either FIR data in binary stream type or TextFIR on text string type. Either one of them can be used without making a difference.

storedFIR:

FIR data to be used as a base. They can be either FIR data in binary stream type or TextFIR in text string type. Either one of them can be used without making a difference.
If this value is Null, FIR data are created based in this value.
If this FIR data include data of right hand thumb, index and middle and capturedFIR data to newly replace include data of right hand thumb and left hand thumb, created FIR data include right hand index and middle in storedFIR data and right hand thumb and left hand thumb in capturedFIR data. That is, new data are created by adding and modifying new data to existing data.

payload:

Payload data to be stored at FIR data to be created. They can be either Payload data in binary stream type or TextPayload data in text string type. Either one of them can be used without making a difference. This value may not be designated as option. If it is not designated, the Payload value of existing data is kept.

Related properties:

ErrorCode, ErrorDescription
FIR, FIRLength, TextFIR

6.5.2. Properties

Various properties of the IFPData interface are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 represents success and all other values represent failure.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored as character string.

It is used to output error values for ErrorCode as character string.

■ **TotalFingerCount**

Prototype:

```
[ReadOnly]  long          TotalFingerCount;
```

Description:

The total number of fingers of converted FIR is stored.

It is used only after the Export method is called.

Related methods:

Export

Related properties:

FingerID

■ **FingerID**

Prototype:

```
[ReadOnly]  long          FingerID(long nIndex);
```

Description:
The finger ID information of converted FIR is stored as array.
nIndex can have a value ranging in 0 ~ (TotalFingerCount – 1).
It is used only after the Export method is called.

Related methods:
Export

Related properties:
FPSampleDataLength, FPSampleData

■ **SampleNumber**

Prototype:

```
[ReadOnly]  long          SampleNumber;
```

Description:
The number of finger-by-finger templates of converted FIR is stored. The value of either 1 or 2 is stored.
It is used only after the Export method is called.

Related methods:
Export

Related properties:
FPSampleDataLength, FPSampleData

■ **FPSampleData**

Prototype:

```
[ReadOnly]  VARIANT       FPSampleData(
                                long nFingerID,
                                long SampleNum);
```

Description:
Binary stream data of finger-by-finger templates of converted FIR are obtained.
nFingerID and SampleNum can be obtained using FingerID and SampleNumber property.
It is used only after the Export method is called.

Parameters:

nFingerID:
The ID of a finger to be obtained.

nSampleNumber:
The number of a sample to be obtained. The value of either 0 or 1 is used.

Related methods:
Export

Related properties:
FingerID, SampleNumber, FPSampleDataLength

■ **FPSampleDataLength**

Prototype:

```
[ReadOnly]  long          FPSampleDataLength(
                                long nFingerID,
                                long SampleNum);
```

Description:
The data size of finger-by-finger template of converted FIR is obtained.
nFingerID and SampleNum can be obtained using FingerID 및 SampleNumber property.
It is used only after the Export method is called.

Parameters:
nFingerID:
The ID of a finger to be obtained.

nSampleNumber:
The number of a sample to be obtained. The value of either 0 or 1 is used.

Related methods:
Export

Related properties:
FingerID, SampleNumber, FPSampleData

■ **FIR**

Prototype:

```
[ReadOnly]  VARIANT       FIR;
```

Description:
FIR data acquired after calling a method such as Import or CreateTemplate are obtained as binary stream data.
The length of data is stored at the FIRLength property.
FIR format to obtain is required to be designated in advance at the FIRFormat property.

Related methods:
Import, CreateTemplate

Related properties:
FIRLength, FIRFormat

■ FIRLength

Prototype:

```
[ReadOnly]  long          FIRLength;
```

Description:
The length of binary stream data acquired after calling a method such as Import or CreateTemplate is obtained.

Related methods:
Import, CreateTemplate

Related properties:
FIR

■ TextFIR

Prototype:

```
[ReadOnly]  BSTR          TextFIR;
```

Description:
FIR data acquired after calling a method such as Import or CreateTemplate are obtained as text string type. Since it is character string, the length is not separately stored.
FIR format to obtain is required to be designated in advance at the FIRFormat property.

Related methods:
Import, CreateTemplate

Related properties:
FIRFormat

■ **FIRFormat**

Prototype:

```
[Read/Write] long        FIRFormat;
```

Description:
The data format of FIR to obtain is designated. When the structure of FIR data is changed later, this format value is changed to maintain the lower compatibility. Only the STANDARD(1) is currently supported.

Related properties:
FIR, TextFIR

6.6. IFPImage Interface

The interface related to a function to obtain fingerprint data as image or save them as image file. To convert fingerprint data into image, this interface is obtained and used.

6.6.1. Methods

Various methods of the IFPImage interface are described.

■ **Export**

Prototype:

```
HRESULT Export ();
```

Description:

Image data are extracted from Audit FIR data acquired most recently. After calling the method, each of image informations can be obtained using various properties.
Since FIR data generally hide internal data through encryption, finger-by-finger image information can not be known. Therefore, to store image finger-by-finger, data need to be converted using this method before use.

Related properties:

ErrorCode, ErrorDescription
TotalFingerCount, SampleNumber, FingerID, RawData, ImageWidth, ImageHeight

Related methods:

IDevice.Capture, IExtraction.Enroll, IExtraction.Verify

■ **ExportEx**

Prototype:

```
HRESULT ExportEx (VARIANT capturedAuditFIR);
```

Description:

Image data are extracted from given Audit FIR data. After calling the method, each of image informations can be obtained using various properties.
FIR data generally hide internal data through encryption, finger-by-finger information can not be known. Therefore, to store image finger-by-finger, data need to be converted using this method before use.
It is equivalent to the Export method. A difference is that the Export method automatically uses Audit FIR data acquired most recently but this method uses given Audit FIR data.

Parameters:

capturedAuditFIR:

Audit FIR data to convert. They can be either Audit FIR data in binary stream type or TextAuditData data in text string type. Either one of them can be used without making a difference.

Related properties:
ErrorCode, ErrorDescription
TotalFingerCount, SampleNumber, FingerID, RawData, ImageWidth, ImageHeight, AuditData, AuditDataLength, TextAuditData

■ **Save**

Prototype:

```
HRESULT Save (
            BSTR bszImgFilePath,
            long nImageType,
            long nFingerID,
            [in,optional] long nSampleNumber);
```

Description:
Exported image data for Audit are saved as file in desired type image format.
nFingerID and nSampleNumber can be obtained using FingerID and SampleNumber property.
This method is used only after the Export or ExportEx method is called.

Parameters:
bszImgFilePath:
An image file to be saved is designated with full path.

nImageType:
The format of an image file to be saved is designated.
Allowed values for use are shown below.

```
1 -       RAW
2 -       BMP
3 -       JPG
```

nFingerID:
The ID of a finger to be saved.

nSampleNumber:
The number of a sample to be saved. The value of either 0 or 1 is used.

If this value is not designated as option, the value of 0 is used.

Related properties:
ErrorCode, ErrorDescription
TotalFingerCount, SampleNumber, FingerID

Related methods:
Export, ExportEx

6.6.2. Properties

Various properties of the IFPImage interface are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 represents success and all other values represent failure.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored as character string.

It is used to output error values for ErrorCode as character string.

■ **TotalFingerCount**

Prototype:

```
[ReadOnly]  long          TotalFingerCount;
```

Description:

The total number of fingers of converted Audit FIR is stored.

It is used only after the Export or ExportEx method is called.

Related methods:

Export, ExportEx

Related properties:

FingerID

■ **FingerID**

Prototype:

```
[ReadOnly]  long          FingerID(long nIndex);
```

Description:
The finger ID information of converted FIR data is stored as array.
nIndex can have a value ranging in 0 ~ (TotalFingerCount – 1).
It is used only after the Export or ExportEx method is called.

Related methods:
Export, ExportEx, Save

Related properties:
RawData

■ **SampleNumber**

Prototype:

```
[ReadOnly]  long          SampleNumber;
```

Description:
The number of finger-by-finger templates of converted Audit FIR is stored. The value of either 1 or 2 is stored.
It is used only after the Export or ExportEx method is called.

Related methods:
Export, ExportEx, Save

Related properties:
RawData

■ **ImageWidth / ImageHeight**

Prototype:

```
[ReadOnly]  long          ImageWidth;
[ReadOnly]  long          ImageHeight;
```

Description:
The image size of converted Audit FIR is stored.
It is used only after the Export or ExportEx method is called.

Related methods:
Export, ExportEx

Related properties:
RawData

■ **RawData**

Prototype:

```
[ReadOnly]  VARIANT      RawData (
                              long nFingerID,
                              [in, optional] long nSampleNumber);
```

Description:
Raw image data of converted Audit FIR are obtained as binary stream.
nFingerID and nSampleNumber can be obtained using FingerID and SampleNumber property.
It is used after the Export or ExportEx method is called.

Related methods:
Export, ExportEx

Related properties:
FingerID, SampleNumber

■ **AuditData**

Prototype:

```
[ReadOnly]  VARIANT      AuditData
```

Description:
Audit FIR data acquired most recently are obtained as binary stream.
Data obtained this way can be used as an argument of the ExportEx method.
To obtain this data, methods such as IDevice.Capture, IExtraction.Enroll, IExtraction.Verify need to be implemented.
The length of binary stream is stored at the AuditDataLength property.

Related methods:
IDevice.Capture, IExtraction.Enroll, IExtraction.Verify, ExportEx

Related properties:
AuditDataLength

■ **AuditDataLength**

Prototype:

```
[ReadOnly]  long          AuditDataLength
```

Description:

The length for binary stream of most recently acquired Audit FIR data is obtained.
To obtain this data, methods such as IDevice.Capture, IExtraction.Enroll, IExtraction.Verify need to be implemented.

Related methods:

IDevice.Capture, IExtraction.Enroll, IExtraction.Verify, ExportEx

Related properties:

AuditData

■ **TextAuditData**

Prototype:

```
[ReadOnly]  BSTR          TextAuditData
```

Description:

More recently acquired Audit FIR data are obtained as character string in text string type.
Data obtained this way can be used as an argument of the ExportEx method.
To obtain this data, methods such as IDevice.Capture, IExtraction.Enroll, IExtraction.Verify need to be implemented.

Related methods:

IDevice.Capture, IExtraction.Enroll, IExtraction.Verify, ExportEx

6.7. IFastSearch Interface

The interface related to FastSearch function to perform 1:N high-speed authentication. To use functions such as FastSearch related DB management and authentication implementation, this interface is obtained and used.

6.7.1. Methods

Various methods of the IFastSearch interface are described.

■ **AddFIR**

Prototype:

```
HRESULT AddFIR (VARIANT FIR, long nUserID);
```

Description:

FIR data are added to the DB for FastSearch.

To implement 1:N authentication, the DB to implement 1:N needs to be built and the DB is created using this function.

Also, 1:N authentication operates in template unit rather than FIR unit. Therefore, even if one FIR is added, several data are added to the DB when several numbers of templates exists inside FIR. Information on added templates can be obtained through a property such as AddedFpInfo.

Parameters:

FIR

FIR data to be added. They can be either FIR data in binary stream type or TextFIR data in text string type. Either one of them can be used without making a difference.

nUserID:

The user ID of FIR to be added.

Related properties:

ErrorCode, ErrorDescription

FpCount, IsFpExisted, AddedFpCount, AddedFpInfo, FpInfo

Related methods:

RemoveFp, RemoveUser, ClearDB, SaveDBToFile, LoadDBFromFile

■ **RemoveFp**

Prototype:

```
HRESULT RemoveFp (long nUserID, long nFingerID, long nSampleNumber);
```

Description:
Specific data are deleted from the DB for FastSearch.

Parameters:
nUserID:
The user ID information of a template to be deleted.

nFingerID:
The finger ID information of a template to be deleted.

nSampleNumber:
The sample number of a template to be deleted.

Related properties:
ErrorCode, ErrorDescription
FpCount, IsFpExisted, FpInfo

Related methods:
AddFIR, RemoveUser, ClearDB, SaveDBToFile, LoadDBFromFile

■ **RemoveUser**

Prototype:

```
HRESULT RemoveUser (long nUserID);
```

Description:
All data for a specific user ID are deleted from the DB for FastSearch.
It is similar to the UCBioAPI_RemoveFpFromFastSearchDB function. There is a difference in that all data are deleted when several numbers of template data for the same user ID are stored at the DB.

Parameters:
nUserID:
The user ID information of a template to be deleted.

Related properties:
ErrorCode, ErrorDescription
FpCount, IsFpExisted, FpInfo

Related methods:

AddFIR, RemoveFp, ClearDB, SaveDBToFile, LoadDBFromFile

■ **ClearDB**

Prototype:

```
HRESULT ClearDB ();
```

Description:

The memory of the DB for FastSearch is released.

Related properties:

ErrorCode, ErrorDescription
FpCount, IsFpExisted, FpInfo

Related methods:

AddFIR, RemoveFp, RemoveUser, SaveDBToFile, LoadDBFromFile

■ **SaveDBToFile**

Prototype:

```
HRESULT SaveDBToFile(BSTR bszFilePath);
```

Description:

The DB for FastSearch is saved as file. If the memory DB is saved as file like this, loading is faster using the LoadDBFromFile method in the next use of the DB.

Parameters:

bszFilePath:
The full path of file name to be saved as file.

Related properties:

ErrorCode, ErrorDescription
FpCount, IsFpExisted, FpInfo

Related methods:

AddFIR, RemoveFp, RemoveUser, ClearDB, LoadDBFromFile

■ **LoadDBFromFile**

Prototype:

```
HRESULT LoadDBFromFile(BSTR bszFilePath);
```

Description:
The DB for FastSearch is loaded to the memory from the file.
Only files saved using the SaveDBToFile method can be loaded this way.

Parameters:
bszFilePath:
The full path of file name to be loaded to the DB.

Related properties:
ErrorCode, ErrorDescription
FpCount, IsFpExisted, FpInfo

Related methods:
AddFIR, RemoveFp, RemoveUser, ClearDB, SaveDBToFile

■ IdentifyUser

Prototype:

```
HRESULT IdentifyUser(VARIANT processedFIR, long nSecuLevel);
```

Description:
1:N authentication with specific FIR is attempted at the DB for FastSearch.
If authentication succeeds after calling the function, authenticated template information can be obtained. Authentication time may vary each time due to the nature of 1:N authentication and the system speed and memory may affect it. Since authentication time may last long, the MaxSearchTime property is used to set the maximum authentication time.

Parameters:
processedFIR:
FIR data to authenticate. They can be either FIR data in binary stream type or TextFIR data in text string type. Either one of them can be used without making a difference.

nSecuLevel:
A security level to use during authentication is designated.
For allowed values, refer to the UCBioAPI_FIR_SECURITY_LEVEL definition.

Related properties:
ErrorCode, ErrorDescription
MatchedFpInfo, MaxSearchTime, UseGroupMatch, MatchMethod

6.7.2. Properties

Various properties of the IFastSearch interface are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 represents success and all other values represent failure.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored as character string.

It is used to output error values for ErrorCode as character string.

■ **FpCount**

Prototype:

```
[ReadOnly]  long          FpCount;
```

Description:

The number of templates stored at the current FastSearch DB is stored.

Related methods:

AddFIR, RemoveFp, RemoveUser, ClearDB, SaveDBToFile, LoadDBFromFile

■ **FpInfo**

Prototype:

```
[ReadOnly]  VARIANT       FpInfo(long index);
```

Description:

Template information stored at the current FastSearch DB is obtained.

This obtained information is obtained with the ITemplateInfo interface.

The index has a value ranging in 0 ~ (FpCount – 1).

Related properties:
ITemplateInfo.UserID, ITemplateInfo.FingerID, ITemplateInfo.SampleNumber

■ **IsFpExisted**

Prototype:

```
[ReadOnly]  BOOL          IsFpExisted(
                                long nUserID,
                                long nFingerID,
                                long nSampleNumber);
```

Description:
It examines if a template with the information designated at the current FastSearch DB exists.

Parameters:
nUserID:
The user ID information of a template to be deleted.

nFingerID:
The finger ID information of a template to be deleted.

nSampleNumber:
The sample number of a template to be deleted.

Related methods:
AddFIR, RemoveFp, RemoveUser, ClearDB, SaveDBToFile, LoadDBFromFile

■ **AddedFpCount**

Prototype:

```
[ReadOnly]  BOOL          AddedFpCount;
```

Description:
When FIR data are added to the FastSearch DB through the AddFIR method, the number of added templates is obtained.
1:N authentication operates internally in template unit rather than in FIR unit. Therefore, even if one FIR is added, several numbers of data are added to the DB when several numbers of templates exist inside FIR. The number of templates added in that way is

stored in this property.
It is used only after the AddFIR is called.

Related methods:
AddFIR

■ **AddedFpInfo**

Prototype:

```
[ReadOnly]  VARIANT      AddedFpInfo(long index);
```

Description:
When FIR data are added to the FastSearch DB through the AddFIR method, each of added template informations is obtained. The obtained information is obtained with the ITemplateInfo interface.
The index can have a value ranging in 0 ~ (AddedFpCount – 1).
It is used only after the AddFIR method is called.

Related methods:
AddFIR

Related properties:
ITemplateInfo.UserID, ITemplateInfo.FingerID, ITemplateInfo.SampleNumber

■ **MatchedFpInfo**

Prototype:

```
[ReadOnly]  VARIANT      MatchedFpInfo;
```

Description:
When 1:N authentication succeeds through the IdentifyUser method, authenticated template information is obtained.
The obtained information is obtained with the ITemplateInfo interface.
It is used only after the IdentifyUser method is called.

Related methods:
IdentifyUser

Related properties:
ITemplateInfo.UserID, ITemplateInfo.FingerID, ITemplateInfo.SampleNumber

■ **MaxSearchTime**

Prototype:

```
[Read/Write] long        MaxSearchTime;
```

Description:

When 1:N authentication is performed through the IdentifyUser method, authentication speed varies depending on the DB size and system performance. To prevent delaying of authentication due to slow authentication, the maximum authentication time can be designated and authentication is performed only within that time period. If authentication is not completed within the designated time, IdentifyUser returns an error.
Since the unit for this value is millisecond, 10,000 is set for 10 seconds.
If this value is 0, it is equivalent to not designating the maximum authentication time. The default value is 0.

Related methods:

IdentifyUser

■ **UseGroupMatch**

Prototype:

```
[Read/Write] long        UseGroupMatch;
```

Description:

When performing 1:N authentication through the IdentifyUser method, it determines if authentication in group unit is performed or not. If it is set as 0, authentication is performed in the order stored at the DB. If is set as 1, authentication in group is performed. The default value is 1.
It is recommended to set this value as 1 for authentication.

Related methods:

IdentifyUser

■ **MatchMethod**

Prototype:

```
[Read/Write] long        MatchMethod;
```

Description:

When performing 1:N authentication through the IdentifyUser method, a method to perform authentication is determined. If it is set as 0, an authentication level is set and

authentication is terminated immediately when it goes over that level. If it is set as 1, it is the highest point authentication method and it searches for the value with the highest authentication level. The default value is 0.
It is recommended to set this value as 0 for authentication.

Related methods:
IdentifyUser

6.8. ITemplateInfo Interface

This interface is used to represent template information in the IFastSearch interface. It consists of only properties without including any method.

6.8.1. Properties

Various properties of the ITemplateInfo interface are described.

■ **UserID**

Prototype:

```
[ReadOnly]  long          UserID;
```

Description:

It has the user ID from the template information of FastSearch.

■ **FingerID**

Prototype:

```
[ReadOnly]  long          FingerID;
```

Description:

It has the finger ID from the template information of FastSearch. Refer to UCBioAPI_FINGER_ID.

■ **SampleNumber**

Prototype:

```
[ReadOnly]  long          SampleNumber;
```

Description:

It has the sample number from the template information of FastSearch. It has the value of either 0 or 1.

6.9. ISmartCard Interface

The interface related to the smart card. To use functions such as storing data to the smart card and loading data, this interface is obtained and used.

- **Note – Functions for smart card use may not include some methods not supported depending on the firmware version of devices.**

6.9.1. Methods

Various methods of the ISmartCard are described.

■ **RFPowerOn**

Prototype:

```
HRESULT RFPowerOn ();
```

Description:

The RF power in the RF range is switched in within 5 seconds.
Most smart card APIs operate with the RF power on.

Related properties:

ErrorCode, ErrorDescription
LED

Related methods:

IDevice.Open, RFPowerOff

■ **RFPowerOff**

Prototype:

```
HRESULT RFPowerOff ();
```

Description:

The RF power in the RF range is switched off within 5 seconds.

Related properties:

ErrorCode, ErrorDescription
LED

Related methods:

RFPowerOn

■ **RFFunction**

Prototype:

```
HRESULT RFFunction(VARIANT CmdBuff, long nCmdLen);
```

Description:

If a card exits in the RF range, a command is transmitted and the result is returned within 5 seconds.
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.

Parameters:

CmdBuff:
The buffer pointer that stores commands to transmit.

nCmdLen:
The length of the buffer pointer that stores commands to transmit.

Related properties:

ErrorCode, ErrorDescription
LED

Related methods:

RFPowerOn

■ **ReadSerial**

Prototype:

```
HRESULT ReadSerial ();
```

Description:

If a card exists in the RF range, the applicable serial number is obtained within 5 seconds.
Since this command includes a function to switch the RF power on, it is not necessary to call the RFPowerOn method before the function call.
After calling the function, the serial number can be obtained through ResultBuffer or Serial property.
(if you wish to receive a string of Result Buffer then you use textSerial property.)

Related properties:

ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength, Serial, textSerial

■ **ReadBlock**

Prototype:

```
HRESULT ReadBlock (long SectorNum, long BlockNum);
```

Description:

If a card exists in the RF range, data in the designated block are obtained within 5 seconds.
The RF power must be switched on.
The AuthMode and Key value appropriate for access rights must be set in advance.
After calling the function, the value can be obtained through ResultBuffer.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to read. The maximum value may vary depending on the memory size of the card.

BlockNum:
The number of a block to read. It has a value ranging in 0 ~ 3.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB, ResultBuffer, ResultLength

Related methods:

RFPowerOn, WriteBlock

■ **WriteBlock**

Prototype:

```
HRESULT WriteBlock (long SectorNum, long BlockNum, VARIANT varData);
```

Description:

If a card exists in the RF range, 16 byte length data are written on the designated block within 5 seconds.
The RF power must be switched on.
The AuthMode and Key value appropriate for access rights must be set in advance.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to be written on. The maximum value may vary depending on the memory size of the card.

BlockNum:
The number of a block to be written on. It has a value ranging in 0 ~ 3.

varData:
16 byte block data to be written on.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB

Related methods:

RFPowerOn, ReadBlock

■ **ReadSector**

Prototype:

```
HRESULT ReadSector (long SectorNum);
```

Description:

If a card exists in the RF range, data in the designated sector are loaded within 5 seconds.
The RF power must be switched on.
The AuthMode and Key value appropriate for access rights must be set in advance.
After calling the function, the value can be obtained through ResultBuffer.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to read. The maximum value may vary depending on the memory size of the card.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB, ResultBuffer, ResultLength

Related methods:

RFPowerOn, WriteSector

■ **WriteSector**

Prototype:

```
HRESULT WriteSector (long SectorNum, VARIANT varData48);
```

Description:

If a card exists in the RF range, 48 byte length data are written on the designated sector within 5 seconds.
The RF power must be switched on.
The AuthMode and Key value appropriate for access rights must be set in advance.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to be written on. The maximum value may vary depending on the memory size of the card.

varData:
48 byte sector data to be written on.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB

Related methods:

RFPowerOn, ReadSector

■ **ReadSectorFieldContent**

Prototype:

```
HRESULT ReadSectorFieldContent (
                    long StartSectorNum,
                    long EndSectorNum);
```

Description:

If a card exists in the RF range, all data in the designated sector range are loaded within 5 seconds.
The maximum range allowed for designation is 10 sectors and the maximum length of data that can be obtained is 480 bytes. (48 Bytes * 10 Sectors = 480 Bytes)
The RF power must be switched on.
The AuthMode and Key value appropriate for access rights must be set in advance.
After calling the function, the value can be obtained through ResultBuffer.
This function is a Mifare card related method.

Parameters:

StartSectorNum:
The starting number of sectors to read. The maximum value may vary depending on the memory size of the card.

EndSectorNum:
The ending number of sectors to read. The maximum value may vary depending on the memory size of the card.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB, ResultBuffer, ResultLength

Related methods:

RFPowerOn, WriteSectorFieldContent

■ **WriteSectorFieldContent**

Prototype:

```
HRESULT WriteSectorFieldContent (
                          long StartSectorNum,
                          long EndSectorNum,
                          VARIANT varData);
```

Description:

If a card exists in the RF range, all data inside the designated sector range are written on within 5 seconds.
The maximum range allowed for designation is 10 sectors and the maximum length of data that can be written on is 480 bytes. (48 Bytes * 10 Sectors = 480 Bytes)
The RF power must be switched on.
The AuthMode and Key value appropriate for access rights must be set in advance.
This function is a Mifare card related method.

Parameters:

StartSectorNum:
The starting number of sectors to be written on. The maximum value may vary depending on the memory size of the card.

EndSectorNum:
The ending number of sectors to be written on. The maximum value may vary depending on the memory size of the card.

varData:
Data to write. The length of data is (EndSectorNum – StartSectorNum + 1) * 48bytes.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB

Related methods:

RFPowerOn, ReadSectorFieldContent

■ **PreValue**

Prototype:

```
HRESULT PreValue (long SectorNum, long BlockNum, long newVal);
```

Description:

5

If a card exists in the RF range, the designated block is switched to the Value mode and 4 byte newVal is written within 5 seconds.

If the pData value to write is 0x00000000, the following value is written on the applicable block after the function call. (0x00000000FFFFFFFF00000000FFFFFFFF00FF00FF)

The block switched to the Value mode this way can use methods for the Value mode such as ReadValue and IncrementValue.

For more detailed information on the Value mode, refer to Using Smart Card in this document.

The RF power must be switched on.

The AuthMode and Key value appropriate for access rights must be set in advance.

This function is a Mifare card related method.

Parameters:

SectorNum:

The number of a sector to be written on. The maximum value may vary depending on the memory size of the card.

BlockNum:

The number of a block to be written on. It has a value ranging in 0 ~ 3.

newVal:

4 byte data to write.

Related properties:

ErrorCode, ErrorDescription

LED, AuthMode, KeyA, KeyB, Value

Related methods:

RFPowerOn, ReadValue, IncrementValue, DecrementValue

■ **ReadValue**

Prototype:

```
HRESULT ReadValue(long SectorNum, long BlockNum);
```

Description:

If a card exists in the RF range, 4 byte Value data in the designated block are loaded within 5 seconds.
The RF power must be switched on.
The AuthMode and Key value appropriate for access rights must be set in advance.
After calling the function, the value can be obtained through the ResultBuffer or Value property.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to read. The maximum value may vary depending on the memory size of the card.

BlockNum:
The number of a block to read. It has a value ranging in 0 ~ 3.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB, ResultBuffer, ResultLength, Value

Related methods:

RFPowerOn, PreValue, IncrementValue, DecrementValue

■ **IncrementValue**

Prototype:

```
HRESULT IncrementValue(long SectorNum, long BlockNum, long newVal);
```

Description:

If a card exists in the RF range, 4 byte Value data in the designated block is increased by a designated value within 5 seconds.
The RF power must be switched on and the designated block must be in the Value mode.
The AuthMode and Key value appropriate for access rights must be set in advance.
After calling the function, the value can be obtained through the ResultBuffer or Value property.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to be written on. The maximum value may vary depending on the memory size of the card.

BlockNum:
The number of a block to be written on. It has a value ranging in 0 ~ 3.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB, ResultBuffer, ResultLength, Value

Related methods:

RFPowerOn, PreValue, ReadValue, DecrementValue

■ **DecrementValue**

Prototype:

```
HRESULT DecrementValue(long SectorNum, long BlockNum, long newVal);
```

Description:

If a card exists in the RF range, 4 byte Value data in the designated block is decreased by a designated value within 5 seconds.
The RF power must be switched on and the designated block must be in the Value mode.
The AuthMode and Key value appropriate for access rights must be set in advance.
After calling the function, the value can be obtained through the ResultBuffer or Value property.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to be written on. The maximum value may vary depending on the memory size of the card.

BlockNum:
The number of a block to be written on. It has a value ranging in 0 ~ 3.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB, ResultBuffer, ResultLength, Value

Related methods:

RFPowerOn, PreValue, ReadValue, IncrementValue

■ **WriteSectorTrailer**

Prototype:

```
HRESULT WriteSectorTrailer (
                    long SectorNum,
                    VARIANT NewAccessBit,
                    VARIANT NewKeyA,
                    VARIANT NewKeyB);
```

Description:

If a card exists in the RF range, the sector trailer area of the designated is modified within 5 seconds.
Key A, access bits and Key B value of the applicable sector are stored at the sector trailer.
The status of success/failure is determined according to access rights of the applicable sector.
The AuthMode and Key value appropriate for access rights must be set in advance.
This function is a Mifare card related method.

Parameters:

SectorNum:
The number of a sector to be written on. The maximum value may vary depending on the memory size of the card.

NewAccessBit:
4 byte length data that stores new access bits.

NewKeyA:
6 byte length data that stores new key A.

NewKeyB:
6 byte length data that stores new key B.

Related properties:

ErrorCode, ErrorDescription
LED, AuthMode, KeyA, KeyB, ResultBuffer, ResultLength, Value

Related methods:

RFPowerOn, PreValue, ReadValue, IncrementValue

■ **ReqA**

Prototype:

```
HRESULT ReqA();
```

Description:

If a type A card exists in the RF range, ATQ is obtained from the card within 5 seconds.
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Related properties:

ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:

RFPowerOn, HaltA

■ **WupA**

Prototype:

```
HRESULT WupA();
```

Description:

If a type A card exists in the RF range, ATQ is obtained from the card within 5 seconds.
It is equivalent to the ReqA method but there is a difference in that a card in halt state also responds.
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Related properties:

ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:

RFPowerOn, HaltA

■ **Select**

Prototype:

```
HRESULT Select();
```

Description:
With ATQ obtained from the type A card, if a type A card exists in the RF range, the SAK and UID value of the card are obtained from the card within 5 seconds.
The length of SAK is 1 byte and the length of UID is variable.
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Related properties:
ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:
RFPowerOn

■ **HaltA**

Prototype:

```
UCBioAPI_RETURN UCBioAPI UCBioAPI_SC_Halt (
              [IN]  UCBioAPI_HANDLE    hHandle,
              [IN]  UCBioAPI_UINT16    wLed);
```

Description:
The type A card in select state is switched to halt state.
The card that became halt state this way does not respond to the ReqA method but responds only to the WupA method.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Related properties:
ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:
RFPowerOn, ReqA, WupA

■ **Rats**

Prototype:

```
HRESULT Rats(long fsdi, long cid);
```

Description:

If a type A card exists in the RF range, the ATS (Answer To Select) value of the card is obtained from the card within 5 seconds. ATS can be obtained when the SAK value is 0x2X (ISO14443-4 supported).
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

Fsdi:

The maximum size of the frame that PCS can receive through the setting of FSDI (Frame Size for proximity coupling Device Integer) can be determined. (Values ranging in 0x00 ~ 0x0F)

cid:

It is possible to call each individual card selectively through CID (Card Identifier). (Values ranging in 0x00 ~ 0x0F)

Related properties:

ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:

RFPowerOn

■ **PpsRequest**

Prototype:

```
HRESULT PpsRequest(long cid, long pps0, long pps1);
```

Description:

If a type A card exists in the RF range, the PPSS value of the card is obtained from the card within 5 seconds.
If parameters that can be changed by the ATS value are supported, it can be used by PCD. That is, if a higher Baud rate is supported at DS and DR, selective parameters in ATS, the Baud rate in both directions can be independently increased by 2, 4, 8 times.
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

cid:
The CID (Card Identifier) value selected during Rats call. (Values ranging in 0x00 ~ 0x0F)

pps0:
It represents if PPS1 is transmitted or not.
It represents transmission if it is 0x11. It represents no transmission if it is 0x01.

pps1:
The upper 2 bytes represent DRI and the lower 2 bytes represent DS.

Related properties:

ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:

RFPowerOn

■ **BlockFormat**

Prototype:

```
HRESULT BlockFormat(
                    long pcb,
                    long CidOrNad,
                    VARIANT inf,
                    long infLen);
```

Description:

If a type A card exists in the RF range, data are loaded from the card within 5 seconds.
The command on block exchange in the data layer (T=1).
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Parameters:

pcb:

It represents PCB (Protocol Control Byte). (Required)
It has information required to control data transmission. It is divided into I – Block, R – Block, S – Block. To read data, I – Block is generally used. The PCB value needs to be transmitted by toggling Bit0 as in 0x0A, 0x0B, 0x0A.

CidOrNad:

The CID or NAD value designated at PICC with Rats. (Optional)

inf:

It represents INF (Information Field).
To read the applicable file, a special format needs to be known.

infLen:

The length of inf.

Related properties:

ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:

RFPowerOn, Deselect

■ **Deselect**

Prototype:

```
HRESULT Deselect(long CidOrNad);
```

Description:

If a type A card in active state exists in the RF range, the card is deactivated within 5 seconds.

If a card in active state is deselected, the card does not respond to BlockFormat as long as it is within the RF range.

After calling the function, the value can be obtained through ResultBuffer.

The RF power must be switched on.

This function is a function related to ISO14443-A.

Parameters:

CidOrNad:

CID or NAD value designated at PICC with Rats.

Related properties:

ErrorCode, ErrorDescription

LED, ResultBuffer, ResultLength

Related methods:

RFPowerOn, BlockFormat

■ **TypeA_ActiveState**

Prototype:

```
HRESULT TypeA_ActiveState();
```

Description:

If a type A card exists in the RF range, ATS is obtained from the card by implementing ReqA, Select and Rats at the same time within 5 seconds.
After calling the function, the value can be obtained through ResultBuffer.
The RF power must be switched on.
This function is a function related to ISO14443-A.

Related properties:

ErrorCode, ErrorDescription
LED, ResultBuffer, ResultLength

Related methods:

RFPowerOn, ReqA, Select, Rats

6.9.2. Properties

Various properties of the ISmartCard interface are described.

■ **ErrorCode**

Prototype:

```
[ReadOnly]  long          ErrorCode;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 represents success and all other values represent failure.

■ **ErrorDescription**

Prototype:

```
[ReadOnly]  BSTR          ErrorDescription;
```

Description:

Values for errors occurred during the setting of executed methods and properties are stored as character string.

It is used to output error values for ErrorCode as character string.

■ **LED**

Prototype:

```
[Read/Write] long         LED;
```

Description:

It designates if the result of success/failure is displayed on the LED of a device or not.

If 1 is set, the LED of a device changes to blue upon success. It changes to red upon failure. If 0 is set, there is no change on the LED.

The default value is 1.

Related methods:

RFFunction, ReadSerial, ReadBlock, WriteBlock, ReadSector, WriteSector, ReadSectorFieldContent, WriteSectorFieldContent, PreValue, ReadValue, IncrementValue, DecrementValue, WriteSectorTrailer, ReqA, WupA, HaltA, Select, Rats, PpsRequest, BlockFormat, Deselect, TypeA_ActiveState

■ **AuthMode**

Prototype:

```
[Read/Write] long        AuthMode;
```

Description:

At a Mifare card related method, it designates which key from key A and key B is used in authentication. To use key A, this value is set as 0x60. To use key B, this value is set as 0x61. The default value is 0x60.

Related methods:

ReadSerial, ReadBlock, WriteBlock, ReadSector, WriteSector, ReadSectorFieldContent, WriteSectorFieldContent, PreValue, ReadValue, IncrementValue, DecrementValue, WriteSectorTrailer

Related properties:

KeyA, KeyB

■ **KeyA / KeyB**

Prototype:

```
[Read/Write] VARIANT     KeyA;
[Read/Write] VARIANT     KeyB;
```

Description:

At a Mifare card related method, the value of either key A or key B to be used is designated.

Either key A or key B is used according to AuthMode.

The key value is a binary array with 6 byte value. The value of "FF FF FF FF FF FF" is stored as the default value.

Related methods:

ReadSerial, ReadBlock, WriteBlock, ReadSector, WriteSector, ReadSectorFieldContent, WriteSectorFieldContent, PreValue, ReadValue, IncrementValue, DecrementValue, WriteSectorTrailer

Related properties:

AuthMode

■ **ResultBuffer**

Prototype:

```
[ReadOnly] VARIANT       ResultBuffer;
```

Description:
After calling a method from smart card methods to read a value, it is the buffer array that stores the result value. It is used only after the method is called. The length of the buffer is stored at the ResultLength property.

Related methods:
ReadSerial, ReadBlock, ReadSector, ReadSectorFieldContent, PreValue, ReadValue, IncrementValue, DecrementValue

Related properties:
ResultLength

■ ResultLength

Prototype:

```
[ReadOnly] long          ResultLength;
```

Description:
After calling a method from smart card methods to read the value, the length of the buffer array that stores the result value is stored. It is used only after the method is called.

Related methods:
ReadSerial, ReadBlock, ReadSector, ReadSectorFieldContent, PreValue, ReadValue, IncrementValue, DecrementValue

Related properties:
ResultBuffer

■ Value

Prototype:

```
[ReadOnly] long          Value;
```

Description:
When reading the Value using a method for Value mode from Mifare card related methods, the value can be obtained through ResultBuffer but the same value can also be obtained through the Value property.

Related methods:
PreValue, ReadValue, IncrementValue, DecrementValue

Related properties:
ResultBuffer

■ **Serial**

Prototype:

```
[ReadOnly] long          Serial;
```

Description:
When reading the serial number using the ReadSerial method from Mifare card related methods, the serial number can be obtained through ResultBuffer but the same number can also be obtained through the Serial property.

Related methods:
ReadSerial

Related properties:
ResultBuffer

7.Distribution Guide

When an application product is developed using UCBioBSP SDK, SDK modules to be included in product distribution are described.

7.1. Common Items

Products developed using UCBioBSP SDK basically require the following modules.

■ **Device driver**

Description:

To use SDK, a device driver appropriate for each device must be installed. But, since a device is not required when only authentication is performed from the server, installation is not necessary.

■ **UCDevice.dll**

Installation folder:

Windows System32 folder (System folder for Win9x)

Description:

To use a device as a device control module of UCBioBSP SDK, this must be installed.

■ **UCBioBSP.dll**

Installation folder:

Windows System32 folder (System folder for Win9x)

Description:

This must be installed as a core module of UCBioBSP SDK.

■ **Skin file**

Installation folder:

Windows System32 folder (System folder for Win9x)

Description:

Required language-by-language skin files are installed if necessary.

7.2. Development Using DLL

When developed using only UCBioBSP.dll, only common items are installed.

7.3. Development Using COM

When developed using UCBioBSPCOM.dll, a COM module, the following files as well as common items are distributed.

■ **UCBioBSPCOM.dll**

Installation folder: Bin32 and Bin64

Module registration: Use is allowed only when the COM module is registered at the Windows registry. Registration is possible through a command "regsvr32 UCBioBSPCOM.dll" or automatic registration is possible using self registration option in the install program.

7.4. Development Using .NET

When developed using UNIONCOMM.SDK.UCBioBSP.dll, a class library for .NET, the following files as well as common items are distributed.

■ **.NET Framework v2.0 or higher**

.NET Framework v2.0 or higher must be installed.

■ **UNIONCOMM.SDK.UCBioBSP.dll**

Installation folder: GAC (Global Assembly Cache) folder

Module registration: Installation is possible using the installation file for .NET included in SDK.

7.5. Development over BioAPI Framework

When developed over the BioAPI Framework, the following files as well as common items are distributed.

■ **BioAPI Framework v2.0 or higher**

BioAPI Framework v2.0 or higher must be installed.

■ **UCBioBSP.dll distribution and registration**

Module registration: UCBioBSP.dll must be registered at the Framework using the BSP registration program provided by the BioAPI Framework before use.

Appendix A. How to enroll fingerprint properly

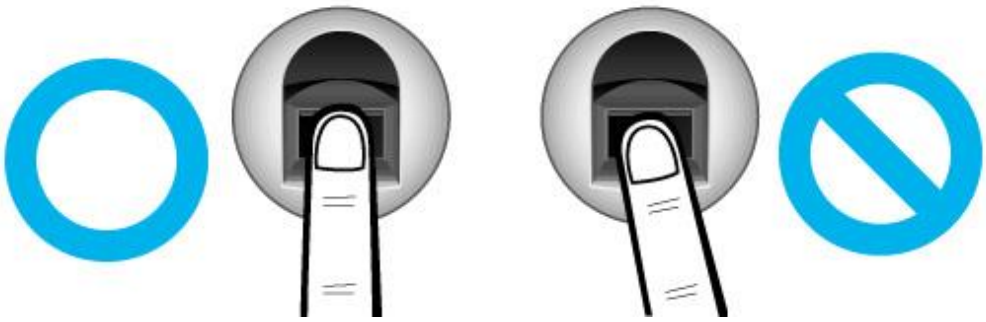
Proper procedures to register a fingerprint to the fingerprint recognition device are described.
Once a fingerprint is registered, it is likely to be used in authentication repeatedly and authentication rate may be reduced significantly by wrongly entered fingerprint data. Therefore, to register a good-quality fingerprint, it is highly recommended to read this appendix carefully.
When distributing a product developed with SDK, this appendix may as well be included in the product to be used as a fingerprint input guide for users.

A.1. Proper Way to Enter Fingerprint

1) Place a finger so that the center of fingerprint is located in the right center of the fingerprint input window.



- Do not press too heard with the finger during input and exert pressure for the finger to barely touch the input window.
- Do not move the finger while the red light is on. Remove the finger after the red light goes off.
- Place the finger in the right direction against the fingerprint input window as shown in the figure.



2) Enter the fingerprint of the index finger if possible.

- Using the index finger if possible, enter the fingerprint as if you are making thumbprint
- Using the index finger enables accurate and stable fingerprint input.

3) Check if the fingerprint is not clear or it is injured.

- It is difficult to recognize too dry or wet fingerprint, unclear fingerprint and injured fingerprint. Register with the fingerprint in a different finger in such a case.



A.2. Improper Fingerprint Input Method

1) When a fingerprint is placed sideways,

- If a fingerprint is placed sideways, the possibility of failure rises.

2) When a fingerprint does not touch the input window well,

- If a fingerprint is detached from the input window, authentication will fail.

3) When the tip of a fingerprint is placed on the input window,

- If the tip of a fingerprint is placed on the input window, authentication will fail.

4) When a fingerprint is turned,

- When a fingerprint is turned or moved, authentication rate will drop.

5) When a fingerprint is placed in the edge of the input window,

- If a fingerprint is placed in the edge of the input window, authentication will fail.

A.3. Procedures to Handle Authentication Failure

1) When a finger is too wet,

- If a finger is too wet, authentication is likely to fail. Enter a fingerprint after drying it.

2) When a finger is too dry (especially in winter),

- Enter a fingerprint after applying adequate moisture (breadth) on the finger.

3) When a finger is injured,

- It is recommended to use a different finger for registration.

4) When foreign substances are on the input window or finger,

- Enter a fingerprint after cleaning the input window or finger well.

A.4. Fingerprint Registration, More Convenient with This Way

1) Register a finger with clear and good fingerprint.

2) Since a child fingerprint may be too small or weak for use, it is necessary to register a new fingerprint every 6 months.

3) If authentication rate is too low or fingerprint is too weak, registering the same finger in multiple times (2~3) enables convenient use.

4) Using the index finger if possible, enter a fingerprint as if you are making thumbprint. Touching the tip of a fingerprint is not a proper way of input. Let the center of a fingerprint touch the fingerprint input window.

A.5. Considerations Depending on the Condition of User Fingerprint

1) It is difficult to recognize too dry or wet fingerprint, unclear fingerprint and injured fingerprint. Register with a different finger in such a case.

2) Preventing a fingerprint from moving or shaking during fingerprint input enables smooth process.

3) Registration may not be possible for seniors with too many fine wrinkles in the fingerprint.

4) Using the index finger enables accurate and stable fingerprint input.

5) It is recommended to register with more than 3 fingerprints.