

SFM SDK

Reference Manual

Version 3.3.0

Jul. 2021

CONTENTS

REVISION NOTES	10
1 INTRODUCTION.....	11
1.1 Contents of the SDK.....	12
2 API COMPATIBILITY	13
3 API SPECIFICATION	20
3.1 Return Codes	20
3.2 Serial Communication API	22
3.2.1 UF_InitCommPort	23
3.2.2 UF_CloseCommPort	24
3.2.3 UF_Reconnect.....	25
3.2.4 UF_SetBaudrate.....	26
3.2.5 UF_SetAsciiMode.....	27
3.3 Socket API.....	28
3.3.1 UF_InitSocket.....	29
3.3.2 UF_CloseSocket.....	30
3.4 Low-Level Packet API.....	31
3.4.1 UF_SendPacket.....	32
3.4.2 UF_SendNetworkPacket.....	34
3.4.3 UF_ReceivePacket	36
3.4.4 UF_ReceiveNetworkPacket	37
3.4.5 UF_SendRawData	38
3.4.6 UF_ReceiveRawData	39
3.4.7 UF_SendDataPacket	40
3.4.8 UF_ReceiveDataPacket	41
3.4.9 UF_SetSendPacketCallback.....	42
3.4.10 UF_SetReceivePacketCallback	43

3.4.11	UF_SetSendDataPacketCallback	44
3.4.12	UF_SetReceiveDataPacketCallback.....	45
3.4.13	UF_SetSendRawDataCallback	46
3.4.14	UF_SetReceiveRawDataCallback.....	47
3.4.15	UF_SetDefaultPacketSize	48
3.4.16	UF_GetDefaultPacketSize.....	49
3.5	Generic Command API	50
3.5.1	UF_Command	51
3.5.2	UF_CommandEx	53
3.5.3	UF_CommandSendData	56
3.5.4	UF_CommandSendDataEx.....	58
3.5.5	UF_Cancel	60
3.5.6	UF_SetProtocol	61
3.5.7	UF_GetProtocol.....	62
3.5.8	UF_GetModuleID	63
3.5.9	UF_SetGenericCommandTimeout	64
3.5.10	UF_SetInputCommandTimeout.....	65
3.5.11	UF_GetGenericCommandTimeout.....	66
3.5.12	UF_GetInputCommandTimeout	67
3.5.13	UF_SetNetworkDelay	68
3.5.14	UF_GetNetworkDelay.....	69
3.6	Module API	70
3.6.1	UF_GetModuleInfo	71
3.6.2	UF_GetModuleString	73
3.6.3	UF_SearchModule.....	74
3.6.4	UF_SearchModuleID	75
3.6.5	UF_SearchModuleBySocket.....	76

3.6.6	UF_SearchModuleIDEx.....	77
3.6.7	UF_CalibrateSensor.....	79
3.6.8	UF_Upgrade	80
3.6.9	UF_Reset	81
3.6.10	UF_Lock.....	82
3.6.11	UF_Unlock	83
3.6.12	UF_ChangePassword	84
3.6.13	UF_PowerOff	85
3.7	System Parameters API.....	86
3.7.1	UF_InitSysParameter	87
3.7.2	UF_GetSysParameter	88
3.7.3	UF_SetSysParameter.....	89
3.7.4	UF_GetMultiSysParameter	90
3.7.5	UF_SetMultiSysParameter	91
3.7.6	UF_Save	92
3.7.7	UF_SaveConfiguration	93
3.7.8	UF_ReadConfigurationHeader	95
3.7.9	UF_LoadConfiguration	96
3.7.10	UF_MakeParameterConfiguration.....	97
3.8	Template Management API.....	98
3.8.1	UF_GetNumOfTemplate.....	100
3.8.2	UF_GetMaxNumOfTemplate.....	101
3.8.3	UF_GetAllUserInfo	102
3.8.4	UF_GetAllUserInfoEx.....	104
3.8.5	UF_SortUserInfo	105
3.8.6	UF_SetUserInfoCallback	106
3.8.7	UF_SetAdminLevel	107

3.8.8	UF_GetAdminLevel.....	108
3.8.9	UF_ClearAllAdminLevel	109
3.8.10	UF_SaveDB.....	110
3.8.11	UF_LoadDB	111
3.8.12	UF_CheckTemplate	112
3.8.13	UF_ReadTemplate	113
3.8.14	UF_ReadOneTemplate	114
3.8.15	UF_SetScanCallback	115
3.8.16	UF_ScanTemplate	116
3.8.17	UF_FixProvisionalTemplate.....	117
3.8.18	UF_SetSecurityLevel	118
3.8.19	UF_GetSecurityLevel	119
3.8.20	UF_SetAuthType	120
3.8.21	UF_GetAuthType	121
3.8.22	UF_GetUserIDByAuthType	122
3.8.23	UF_ResetAllAuthType.....	123
3.8.24	UF_SetEntranceLimit.....	124
3.8.25	UF_GetEntranceLimit	125
3.8.26	UF_ClearAllEntranceLimit	126
3.9	Image Manipulation API.....	127
3.9.1	UF_ConvertToBitmap	128
3.9.2	UF_SaveImage	130
3.9.3	UF_LoadImage	131
3.9.4	UF_ReadImage.....	132
3.9.5	UF_ScanImage	133
3.10	Enroll API	134
3.10.1	UF_Enroll.....	135

3.10.2	UF_EnrollContinue.....	139
3.10.3	UF_EnrollAfterVerification.....	141
3.10.4	UF_EnrollTemplate.....	143
3.10.5	UF_EnrollMultipleTemplates	144
3.10.6	UF_EnrollImage.....	145
3.10.7	UF_SetEnrollCallback.....	146
3.11	Identify API.....	147
3.11.1	UF_Identify	148
3.11.2	UF_IdentifyTemplate	149
3.11.3	UF_IdentifyImage	150
3.11.4	UF_SetIdentifyCallback	151
3.12	Verify API.....	152
3.12.1	UF_Verify	153
3.12.2	UF_VerifyTemplate	154
3.12.3	UF_VerifyHostTemplate	155
3.12.4	UF_VerifyImage	156
3.12.5	UF_SetVerifyCallback	157
3.13	Delete API.....	158
3.13.1	UF_Delete	159
3.13.2	UF_DeleteOneTemplate	160
3.13.3	UF_DeleteMultipleTemplates.....	161
3.13.4	UF_DeleteAll.....	162
3.13.5	UF_DeleteAllAfterVerification.....	163
3.13.6	UF_SetDeleteCallback	164
3.14	IO API for SFM3500.....	165
3.14.1	UF_InitIO.....	166
3.14.2	UF_SetInputFunction.....	167

3.14.3	UF_GetInputFunction	169
3.14.4	UF_GetInputStatus.....	170
3.14.5	UF_GetOutputEventList.....	171
3.14.6	UF_ClearAllOutputEvent	174
3.14.7	UF_ClearOutputEvent.....	175
3.14.8	UF_SetOutputEvent	176
3.14.9	UF_GetOutputEvent.....	178
3.14.10	UF_SetOutputStatus	179
3.14.11	UF_SetLegacyWiegandConfig(Deprecated).....	180
3.14.12	UF_GetLegacyWiegandConfig(Deprecated)	181
3.14.13	UF_MakeIOConfiguration	182
3.15	GPIO API for SFM3000/4000/5000/6000	183
3.15.1	UF_GetGPIOConfiguration	184
3.15.2	UF_SetInputGPIO	186
3.15.3	UF_SetOutputGPIO	188
3.15.4	UF_SetSharedGPIO.....	189
3.15.5	UF_DisableGPIO	190
3.15.6	UF_ClearAllGPIO.....	191
3.15.7	UF_SetDefaultGPIO.....	192
3.15.8	UF_EnableWiegandInput	193
3.15.9	UF_EnableWiegandOutput	194
3.15.10	UF_DisableWiegandInput	195
3.15.11	UF_DisableWiegandOutput	196
3.15.12	UF_MakeGPIOConfiguration	197
3.16	User Memory API	198
3.16.1	UF_WriteUserMemory.....	199
3.16.2	UF_ReadUserMemory	200

3.17	Log Management API	201
3.17.1	UF_SetTime	203
3.17.2	UF_GetTime	204
3.17.3	UF_GetNumOfLog.....	205
3.17.4	UF_ReadLog.....	206
3.17.5	UF_ReadLatestLog	207
3.17.6	UF_DeleteOldestLog.....	208
3.17.7	UF_DeleteAllLog	209
3.17.8	UF_ClearLogCache	210
3.17.9	UF_ReadLogCache	211
3.17.10	UF_SetCustomLogField.....	212
3.17.11	UF_GetCustomLogField	213
3.18	Extended Wiegand API	214
3.18.1	UF_SetWiegandFormat.....	215
3.18.2	UF_GetWiegandFormat	217
3.18.3	UF_SetWiegandIO	218
3.18.4	UF_GetWiegandIO.....	219
3.18.5	UF_SetWiegandOption	220
3.18.6	UF_GetWiegandOption	221
3.18.7	UF_SetAltValue	222
3.18.8	UF_ClearAltValue	223
3.18.9	UF_GetAltValue	224
3.18.10	UF_MakeWiegandConfiguration	225
3.19	Wiegand Command Card API.....	226
3.19.1	UF_AddWiegandCommandCard	227
3.19.2	UF_GetWiegandCommandCardList.....	229
3.19.3	UF_ClearAllWiegandCommandCard.....	230

3.20	SmartCard API	231
3.20.1	UF_ReadSmartCard	232
3.20.2	UF_ReadSmartCardWithAG	234
3.20.3	UF_WriteSmartCard	235
3.20.4	UF_WriteSmartCardWithAG	237
3.20.5	UF_WriteSmartCardWithEntranceLimit	239
3.20.6	UF_FormatSmartCard	241
3.20.7	UF_SetSmartCardMode	242
3.20.8	UF_GetSmartCardMode	243
3.20.9	UF_ChangePrimaryKey	244
3.20.10	UF_ChangeSecondaryKey	245
3.20.11	UF_SetKeyOption	246
3.20.12	UF_GetKeyOption	247
3.20.13	UF_SetCardLayout	248
3.20.14	UF_GetCardLayout	249
3.20.15	UF_SetSmartCardCallback	250
3.21	Access Control API	251
3.21.1	UF_AddTimeSchedule	252
3.21.2	UF_GetTimeSchedule	254
3.21.3	UF_DeleteTimeSchedule	255
3.21.4	UF_DeleteAllTimeSchedule	256
3.21.5	UF_AddHoliday	257
3.21.6	UF_GetHoliday	259
3.21.7	UF_DeleteHoliday	260
3.21.8	UF_DeleteAllHoliday	261
3.21.9	UF_AddAccessGroup	262
3.21.10	UF_GetAccessGroup	263

3.21.11	UF_DeleteAccessGroup	264
3.21.12	UF_DeleteAllAccessGroup	265
3.21.13	UF_SetUserAccessGroup	266
3.21.14	UF_GetUserAccessGroup	267
3.22	Blacklist API	268
3.22.1	UF_AddBlacklist	269
3.22.2	UF_DeleteBlacklist	270
3.22.3	UF_GetBlacklist	271
3.22.4	UF_DeleteAllBlacklist	272
3.23	WSQ API	273
3.23.1	UF_ScanImageEX	274
3.23.2	UF_ReadImageEx	275
3.23.3	UF_WSQ_Decode	276
3.24	File System API	278
3.24.1	UF_ResetSystemConfiguration	279
3.24.2	UF_FormatUserDatabase	280
3.25	Secure Packet Protocol API	281
3.25.1	UF_GetSecurePacketProtocolMode	282
3.25.2	UF_SetSecurePacketProtocolMode	283
3.25.3	UF_SetSecureCode	284
3.25.4	UF_CreateKeyPair	285
3.25.5	UF_GetSecureKey	286
3.25.6	UF_PublicKeyExchange	287
3.26	Key Management API	288
3.26.1	UF_ChangeKey	289
4	CONTACT INFO	291

Revision Notes

2021-07-14

- Released v3.3.0
- Added Secure Packet Protocol APIs
- Added Key Management API.

2021-06-07

- o Document style has been changed.

1 Introduction

The SFM SDK is a collection of APIs for interfacing with SFM modules. In addition to simple wrapper functions for Packet Protocol, it also provides high level APIs such as template DB management, image manipulation, etc. By using the SDK, developers could write applications quickly on various development environment without knowing the minute details of Packet Protocol.

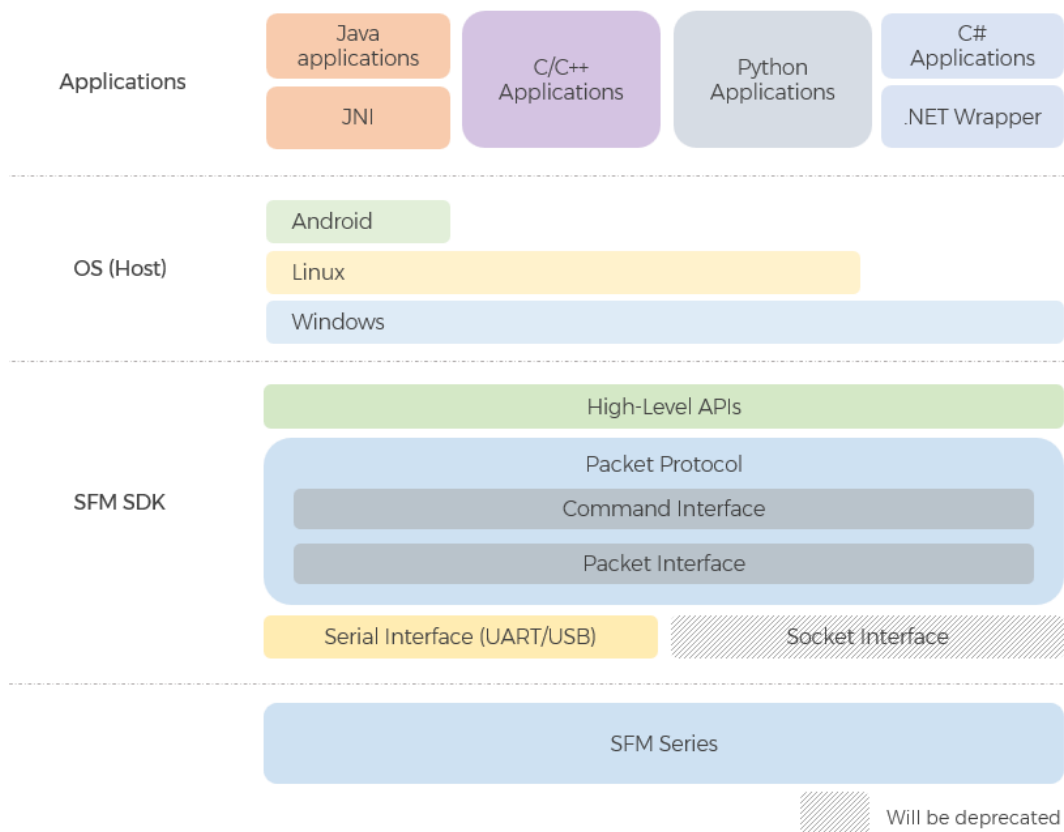


Figure 1. SFM SDK

As shown in Figure 1., the SDK is composed of several layers and developers could choose whichever layer suited for their applications. Another strong point of the SDK is its extensibility. Many of core APIs provide callback mechanism, with which developers can add customized functions. UniFingerUI that released as a demo software is a good example of this feature. The UniFingerUI covers all the core functionalities of SFM modules and shows how to use the SDK in real applications. The source codes of it are also provided in the SDK.

1.1 Contents of the SDK

Documents

- SFM SDK Reference Manual
- SFM Series Packet Protocol Manual

Libraries

By default, the following files are provided, but some files may be added.

- C/C++
 - Header files
 - SFM_SDK.dll (Windows) / libSFM_SDK.so (Linux)
- C#
 - SFM_SDK.dll
 - SFM_SDK_NET.dll (Wrapper DLL)
- Java
 - SFM_SDK.dll (Windows) / libSFM_SDK.so (Linux)
 - SFM_SDK_JNI.dll (Windows) / libSFM_SDK_JNI.so (Linux)
 - SFM_SDK_JNI.jar (JAR)
- Python
 - PyPI (Python Package Index)
 - <https://pypi.org/project/pysfm/>
- Android
 - JitPack
 - <https://jitpack.io/#supremainc/sfm-sdk-android>

For more details,

Please visit SFM Developer Center. You can access all information about the SDK at the SFM Developer Center. (<https://sfm.supremainc.com/>)

2 API Compatibility

Category	Function	SFM 3000 /5000	SFM 4000	SFM 3500	SFM 5500	SFM 6000 / Slim
Serial Comm. API	UF_InitCommPort	○	○	○	○	○
	UF_CloseCommPort	○	○	○	○	○
	UF_Reconnect	○	○	○	○	○
	UF_SetBaudrate	○	○	○	○	○
	UF_SetAsciiMode	○	○	○	○	○
Socket API	UF_InitSocket	○	○	○	○	○
	UF_CloseSocket	○	○	○	○	○
Low-Level Packet API	UF_SendPacket	○	○	○	○	○
	UF_ReceivePacket	○	○	○	○	○
	UF_SendNetworkPacket	○	○	○	○	○
	UF_ReceiveNetworkPacket	○	○	○	○	○
	UF_SendRawData	○	○	○	○	○
	UF_ReceiveRawData	○	○	○	○	○
	UF_SendDataPacket	○	○	○	○	○
	UF_SetSendPacketCallback	○	○	○	○	○
	UF_SetReceivePacketCallback	○	○	○	○	○
	UF_SetSendDataPacketCallback	○	○	○	○	○
	UF_SetReceiveDataPacketCallback	○	○	○	○	○
	UF_SetSendRawDataCallback	○	○	○	○	○
	UF_SetReceiveRawDataCallback	○	○	○	○	○
	UF_SetDefaultPacketSize	○	○	○	○	○
	UF_GetDefaultPacketSize	○	○	○	○	○
General Command API	UF_Command	○	○	○	○	○
	UF_CommandEx	○	○	○	○	○
	UF_CommandSendData	○	○	○	○	○
	UF_CommandSendDataEx	○	○	○	○	○
	UF_Cancel	○	○	○	○	○
	UF_SetProtocol	○	○	○	○	○

	UF_GetProtocol	○	○	○	○	○
	UF_GetModuleID	○	○	○	○	○
	UF_SetGenericCommandTimeout	○	○	○	○	○
	UF_SetInputCommandTimeout	○	○	○	○	○
	UF_GetGenericCommandTimeout	○	○	○	○	○
	UF_GetInputCommandTimeout	○	○	○	○	○
	UF_SetNetworkDelay	○	○	○	○	○
	UF_GetNetworkDelay	○	○	○	○	○
Module API	UF_GetModuleInfo	○	○	○	○	○
	UF_GetModuleString	○	○	○	○	○
	UF_SearchModule	○	○	○	○	○
	UF_SearchModuleID	○	○	○	○	○
	UF_SearchModuleBySocket	○	○	○	○	○
	UF_SearchModuleIDEx	○	○	○	○	○
	UF_CalibrateSensor	○	○	○	○	○
	UF_Upgrade	○	○	○	○	○
	UF_Reset	○	○	○	○	○
	UF_PowerOff	X	○	X	X	X
	UF_Lock	X	X	○	○	X
	UF_Unlock	X	X	○	○	X
	UF_ChangePassword	X	X	○	○	X
System Parameters API	UF_InitSysParameter	○	○	○	○	○
	UF_GetSysParameter	○	○	○	○	○
	UF_SetSysParameter	○	○	○	○	○
	UF_GetMultiSysParameter	○	○	○	○	○
	UF_SetMultiSysParameter	○	○	○	○	○
	UF_Save	○	○	○	○	○
	UF_SaveConfiguration	○	○	○	○	○
	UF_ReadConfigurationHeader	○	○	○	○	○
	UF_LoadConfiguration	○	○	○	○	○
	UF_MakeParameterConfiguration	○	○	○	○	○
Template	UF_GetNumOfTemplate	○	○	○	○	○

Mgmt. API	UF_GetMaxNumOfTemplate	○	○	○	○	○
	UF_GetAllUserInfo	○	○	○	○	○
	UF_GetAllUserInfoEx	X	X	X	X	X
	UF_SortUserInfo	○	○	○	○	○
	UF_SetUserInfoCallback	○	○	○	○	○
	UF_SetAdminLevel	○	○	○	○	○
	UF_GetAdminLevel	○	○	○	○	○
	UF_ClearAllAdminLevel	○	○	○	○	○
	UF_SaveDB	○	○	○	○	○
	UF_LoadDB	○	○	○	○	○
	UF_CheckTemplate	○	○	○	○	○
	UF_ReadTemplate	○	○	○	○	○
	UF_ReadOneTemplate	○	○	○	○	○
	UF_SetScanCallback	○	○	○	○	○
	UF_ScanTemplate	○	○	○	○	○
	UF_FixProvisionalTemplate	○	○	○	○	○
	UF_SetSecurityLevel	○	○	○	○	○
	UF_GetSecurityLevel	○	○	○	○	○
	UF_SetAuthType*	X	X	○	○	X
	UF_GetAuthType*	X	X	○	○	X
	UF_GetUserIDByAuthType*	X	X	○	○	X
	UF_ResetAllAuthType*	X	X	○	○	X
	UF_SetEntranceLimit*	X	X	○	○	X
	UF_GetEntranceLimit*	X	X	○	○	X
	UF_ClearAllEntranceLimit*	X	X	○	○	X
Image API	UF_ConvertToBitmap	○	○	○	○	○
	UF_SaveImage	○	○	○	○	○
	UF_LoadImage	○	○	○	○	○
	UF_ReadImage	○	○	○	○	○
	UF_ScanImage	○	○	○	○	○
Enroll API	UF_Enroll	○	○	○	○	○
	UF_EnrollContinue	○	○	○	○	○

	UF_EnrollAfterVerification	○	○	○	○	○
	UF_EnrollTemplate	○	○	○	○	○
	UF_EnrollMultipleTemplates	○	○	○	○	○
	UF_EnrollImage	○	○	○	○	○
	UF_SetEnrollCallback	○	○	○	○	○
Identify API	UF_Identify	○	○	○	○	○
	UF_IdentifyTemplate	○	○	○	○	○
	UF_IdentifyImage	○	○	○	○	○
	UF_SetIdentifyCallback	○	○	○	○	○
Verify API	UF_Verify	○	○	○	○	○
	UF_VerifyTemplate	○	○	○	○	○
	UF_VerifyHostTemplate	○	○	○	○	○
	UF_VerifyImage	○	○	○	○	○
	UF_SetVerifyCallback	○	○	○	○	○
Delete API	UF_Delete	○	○	○	○	○
	UF_DeleteOneTemplate	○	○	○	○	○
	UF_DeleteMultipleTemplates	○	○	○	○	○
	UF_DeleteAll	○	○	○	○	○
	UF_DeleteByScan	○	○	○	○	○
	UF_SetDeleteCallback	○	○	○	○	○
I/O API	UF_InitIO	X	X	○	○	X
	UF_SetInputFunction	X	X	○	○	X
	UF_GetInputFunction	X	X	○	○	X
	UF_GetInputStatus	X	X	○	○	X
	UF_GetOutputEventList	X	X	○	○	X
	UF_ClearAllOutputEvent	X	X	○	○	X
	UF_ClearOutputEvent	X	X	○	○	X
	UF_SetOutputEvent	X	X	○	○	X
	UF_GetOutputEvent	X	X	○	○	X
	UF_SetOutputStatus	X	X	○	○	X
	UF_SetLegacyWiegandConfig	X	X	○	○	X
	UF_GetLegacyWiegandConfig	X	X	○	○	X

	UF_MakeIOConfiguration	X	X	O	O	X
GPIO API	UF_GetGPIOConfiguration	O	O	X	X	O
	UF_SetInputGPIO	O	O	X	X	O
	UF_SetOutputGPIO	O	O	X	X	O
	UF_SetSharedGPIO	O	O	X	X	O
	UF_DisableGPIO	O	O	X	X	O
	UF_ClearAllGPIO	O	O	X	X	O
	UF_SetDefaultGPIO	O	O	X	X	O
	UF_EnableWiegandInput	O	O	X	X	O
	UF_EnableWiegandOutput	O	O	X	X	O
	UF_DisableWiegandInput	O	O	X	X	O
	UF_DisableWiegandOutput	O	O	X	X	O
	UF_MakeGPIOConfiguration	O	O	X	X	O
User Memory API	UF_WriteUserMemory	O	O	O	O	O
	UF_ReadUserMemory	O	O	O	O	O
Log and Time Mgmt. API	UF_SetTime	X	X	O	O	X
	UF_GetTime	X	X	O	O	X
	UF_GetNumOfLog	X	X	O	O	X
	UF_ReadLog	X	X	O	O	X
	UF_ReadLatestLog	X	X	O	O	X
	UF_DeleteOldestLog	X	X	O	O	X
	UF_DeleteAllLog	X	X	O	O	X
	UF_ClearLogCache	X	X	O	O	X
	UF_ReadLogCache	X	X	O	O	X
	UF_SetCustomLogField	X	X	O	O	X
	UF_GetCustomLogField	X	X	O	O	X
Extended Wiegand API	UF_SetWiegandFormat	X	X	O	O	X
	UF_GetWiegandFormat	X	X	O	O	X
	UF_SetWiegandIO	X	X	O	O	X
	UF_GetWiegandIO	X	X	O	O	X
	UF_SetWiegandOption	X	X	O	O	X
	UF_GetWiegandOption	X	X	O	O	X

	UF_SetAltValue	X	X	O	O	X
	UF_ClearAltValue	X	X	O	O	X
	UF_GetAltValue	X	X	O	O	X
	UF_MakeWiegandConfiguration	X	X	O	O	X
Wiegand Command Card API	UF_AddWiegandCommandCard	X	X	O	O	X
	UF_GetWiegandCommandCardList	X	X	O	O	X
	UF_ClearAllWiegandCommandCard	X	X	O	O	X
SmartCard API	UF_ReadSmartCard	X	X	X	X	X
	UF_ReadSmartCardWithAG	X	X	X	X	X
	UF_WriteSmartCard	X	X	X	X	X
	UF_WriteSmartCardWithAG	X	X	X	X	X
	UF_WriteSmartCardWithEntranceLimit*	X	X	X	X	X
	UF_FormatSmartCard	X	X	X	X	X
	UF_SetSmartCardMode	X	X	X	X	X
	UF_GetSmartCardMode	X	X	X	X	X
	UF_ChangePrimaryKey	X	X	X	X	X
	UF_ChangeSecondaryKey	X	X	X	X	X
	UF_SetKeyOption	X	X	X	X	X
	UF_GetKeyOption	X	X	X	X	X
	UF_SetCardLayout	X	X	X	X	X
	UF_GetCardLayout	X	X	X	X	X
	UF_SetSmartCardCallback	X	X	X	X	X
Access Control API	UF_AddTimeSchedule	X	X	X	X	X
	UF_GetTimeSchedule	X	X	X	X	X
	UF_DeleteTimeSchedule	X	X	X	X	X
	UF_DeleteAllTimeSchedule	X	X	X	X	X
	UF_AddHoliday	X	X	X	X	X
	UF_GetHoliday	X	X	X	X	X
	UF_DeleteHoliday	X	X	X	X	X
	UF_DeleteAllHoliday	X	X	X	X	X
	UF_AddAccessGroup	X	X	X	X	X
	UF_GetAccessGroup	X	X	X	X	X

	UF_DeleteAccessGroup	X	X	X	X	X
	UF_DeleteAllAccessGroup	X	X	X	X	X
	UF_SetUserAccessGroup	X	X	X	X	X
	UF_GetUserAccessGroup	X	X	X	X	X
Blacklist API	UF_AddBlacklist	X	X	O	O	X
	UF_DeleteBlacklist	X	X	O	O	X
	UF_GetBlacklist	X	X	O	O	X
	UF_DeleteAllBlacklist	X	X	O	O	X
WSQ API	UF_ScanImageEx	X	X	X	O	O
	UF_ReadImageEx	X	X	X	O	O
	UF_WSQ_Decode	X	X	X	O	O
File System API	UF_ResetSystemConfiguration	X	X	X	X	O
	UF_FormatUserDatabase	X	X	X	X	O
Secure Packet Protocol API	UF_GetSecurePacketProtocolMode	X	X	X	X	O
	UF_SetSecurePacketProtocolMode	X	X	X	X	O
	UF_SetSecureCode	X	X	X	X	O
	UF_CreateRandomSecureKey	X	X	X	X	O
	UF_CreateKeyPair	X	X	X	X	O
	UF_GetSecureKey	X	X	X	X	O
	UF_PublicKeyExchange	X	X	X	X	O
Key Mgmt. API	UF_ChangeKey	X	X	X	X	O

3 API Specification

3.1 Return Codes

Most APIs in the SDK return UF_RET_CODE. The return codes and their meanings are as follows;

Category	Code	Description
Success	UF_RET_SUCCESS	The function succeeds.
Serial Comm.	UF_ERR_CANNOT_OPEN_SEREIAL	Cannot open the specified serial port.
	UF_ERR_CANNOT_SETUP_SERIAL	Cannot set the baud rate.
	UF_ERR_CANNOT_WRITE_SERIAL	Cannot write data to the serial port.
	UF_ERR_WRITE_SERIAL_TIMEOUT	Write timeout.
	UF_ERR_CANNOT_READ_SERIAL	Cannot read data from the serial port.
	UF_ERR_READ_SERIAL_TIMEOUT	Read timeout.
	UF_ERR_CHECKSUM_ERROR	Received packet has wrong checksum.
	UF_ERR_CANNOT_SET_TIMEOUT	Cannot set communication timeout.
Socket	UF_ERR_CANNOT_START_SOCKET	Cannot initialize the socket interface.
	UF_ERR_CANNOT_OPEN_SOCKET	Cannot open the socket.
	UF_ERR_CANNOT_CONNECT_SOCKET	Cannot connect to the socket.
	UF_ERR_CANNOT_READ_SOCKET	Cannot read data from the socket.
	UF_ERR_READ_SOCKET_TIMEOUT	Read timeout.
	UF_ERR_CANNOT_WRITE_SOCKET	Cannot write data to the socket.
	UF_ERR_WRITE_SOCKET_TIMEOUT	Write timeout.
Protocol	UF_ERR_SCAN_FAIL	Sensor or fingerprint input has failed.
	UF_ERR_NOT_FOUND	Identification failed, or the requested data is not found.
	UF_ERR_NOT_MATCH	Fingerprint does not match.
	UF_ERR_TRY_AGAIN	Fingerprint image is not good.
	UF_ERR_TIME_OUT	Timeout for fingerprint input.

	UF_ERR_MEM_FULL	No more templates are allowed.
	UF_ERR_EXIST_ID	The specified user ID already exists.
	UF_ERR_FINGER_LIMIT	The number of fingerprints enrolled in same ID exceeds its limit.
	UF_ERR_UNSUPPORTED	The command is not supported.
	UF_ERR_INVALID_ID	The requested user ID is invalid or missing.
	UF_ERR_TIMEOUT_MATCH	Timeout for fingerprint identification.
	UF_ERR_BUSY	Module is processing another command.
	UF_ERR_CANCELED	The command is canceled.
	UF_ERR_DATA_ERROR	The checksum of a data packet is incorrect.
	UF_ERR_EXIST_FINGER	The finger is already enrolled.
	UF_ERR_DURESS_FINGER	A duress finger is detected.
	UF_ERR_CARD_ERROR	Cannot read a smart card.
	UF_ERR_LOCKED	Module is locked.
	UF_ERR_ACCESS_NOT_GRANTED	Access is not granted by time schedule and access group.
	UF_ERR_REJECTED_ID	Authentication type of the user is UF_AUTH_REJECT or the ID is in the blacklist.
	UF_ERR_FAKE_DETECTED	Scanned finger is determined as a fake finger.
Application	UF_ERR_EXCEED_ENTRANCE_LIMIT	Entrance limit is exceeded.
	UF_ERR_OUT_OF_MEMORY	Out of memory.
	UF_ERR_INVALID_PARAMETER	Invalid parameter.
	UF_ERR_FILE_IO	File I/O failed
	UF_ERR_INVALID_FILE	The configuration or DB file is invalid.
	UF_ERR_RECOVERY_MODE	Module is failed to start booting.
	UF_ERR_NO_SERIAL_NUMBER	Module has no serial number.
	UF_ERR_INVALID_DATABASE_FORMAT	The format of the DB file is invalid.
	UF_ERR_SECURE_CODE_VERIFICATION_FAIL	Cannot verify the secure code for the secure packet protocol.

3.2 Serial Communication API

To communicate with SFM modules, users should configure the serial port first.

- UF_InitCommPort: configures serial port parameters.
- UF_CloseCommPort: closes the serial port.
- UF_Reconnect: resets system parameters and IO settings.
- UF_SetBaudrate: changes the baud rate.
- UF_SetAsciiMode: changes the packet translation mode.

3.2.1 UF_InitCommPort

Opens a serial port and configures communication parameters. This function should be called first before calling any other APIs.

Function Prototype

```
UF_RET_CODE UF_InitCommPort( const char* commPort, int baudrate, BOOL
asciiMode );
```

Parameters

commPort

Pointer to a null-terminated string that specifies the name of the serial port.

baudrate

Specifies the baud rate at which the serial port operates. Available baud rates are 9600, 19200, 38400, 57600, 115200bps (230400bps and 460800bps are available in later version of 5.0). The default setting of SFM modules is 115200bps.

asciiMode

Determines the packet translation mode. If it is set to TRUE, the binary packet is converted to ASCII format first before being sent to the module. Response packets are in ASCII format, too. The default setting of SFM modules is binary mode.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
UF_RET_CODE result = UF_InitCommPort( "COM1", 115200, FALSE );
```


3.2.2 UF_CloseCommPort

Closes the serial port opened by [UF_InitCommPort](#).

Function Prototype

```
UF_RET_CODE UF_CloseCommPort();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.2.3 UF_Reconnect

To improve communication efficiency, the SDK caches basic information of a module such as system parameters and I/O settings. UF_Reconnect clears this cached information. When changing the modules connected to the serial port, this function should be called.

Function Prototype

```
void UF_Reconnect();
```

Parameters

None

Return Values

None

3.2.4 UF_SetBaudrate

Changes the baud rate.

Function Prototype

```
UF_RET_CODE UF_SetBaudrate( int baudrate );
```

Parameters

baudrate

Specifies the baud rate at which the serial port operates. Available baud rates are 9600, 19200, 38400, 57600, 115200bps. The default setting of SFM modules is 115200bps.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.2.5 UF_SetAsciiMode

Changes the packet translation mode.

Function Prototype

```
void UF_SetAsciiMode( BOOL asciiMode );
```

Parameters

asciiMode

TRUE for ascii format, FALSE for binary format.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.3 Socket API

In addition to serial ports, users can connect to the module by Ethernet-to-Serial converters. In this case, socket API should be used in place of serial API.

- UF_InitSocket: opens a socket and connects to the specified IP address.
- UF_CloseSocket: closes the socket.

3.3.1 UF_InitSocket

Initializes the socket interface and connects to the module with specified IP address.

Function Prototype

```
UF_RET_CODE UF_InitSocket( const char* inetAddr, int port, BOOL  
asciiMode );
```

Parameters

inetAddr

IP address of the Ethernet-to-Serial converter.

port

TCP port of the socket interface.

asciiMode

Determines the packet translation mode. If it is set to TRUE, the binary packet is converted to ASCII format first before being sent to the module. Response packets are in ASCII format, too. The default setting of SFM modules is binary mode.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
UF_RET_CODE result = UF_InitSocket( "192.168.1.41", 1470, FALSE );
```

3.3.2 UF_CloseSocket

Closes the socket interface.

Function Prototype

```
UF_RET_CODE UF_CloseSocket();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4 Low-Level Packet API

These functions provide direct interface to the low-level packet exchanges. In most cases, users need not to call these functions directly. Command API and other high level APIs are implemented on top of this API.

Packet API also let users set callback functions for data transfer. Examples of using these callback functions for GUI application can be found in UniFingerUI V4.0 source codes.

- UF_SendPacket: sends a 13 byte packet.
- UF_SendNetworkPacket: sends a 15 byte network packet.
- UF_ReceivePacket: receives a 13 byte packet.
- UF_ReceiveNetworkPacket: receives a 15 byte network packet.
- UF_SendRawData: sends raw data.
- UF_ReceiveRawData: receives raw data.
- UF_SendDataPacket: sends data using Extended Data Transfer Protocol.
- UF_ReceiveDataPacket: receives data using Extended Data Transfer Protocol.
- UF_SetSendPacketCallback: sets the callback function of sending packets.
- UF_SetReceivePacketCallback: sets the callback function of receiving packets.
- UF_SetSendDataPacketCallback: sets the callback function of sending data packets.
- UF_SetReceiveDataPacketCallback: sets the callback function of receiving data packets.
- UF_SetSendRawDataCallback: sets the callback function of sending raw data.
- UF_SetReceiveRawDataCallback: sets the callback function of receiving raw data.
- UF_SetDefaultPacketSize: sets the size of data packets.
- UF_GetDefaultPacketSize: gets the size of data packets.

3.4.1 UF_SendPacket

Sends a 13 byte packet to the module. The packet is composed as follows;

Start code	Command	Param	Size	Flag/Error	Checksum	End code
1byte	1byte	4bytes	4bytes	1byte	1byte	1byte

- Start code: 1 byte. Indicates the beginning of a packet. It always should be 0x40.
- Command: 1 byte. Refer to the *Packet Protocol Manual* for available commands.
- Param: 4 bytes. The meaning of this field varies according to each command.
- Size: 4 bytes. The meaning of this field varies according to each command.
- Flag/Error: 1 byte. Indicates flag data in the request packet, and error code in the response packet.
- Checksum: 1 byte. Checks the validity of a packet. Checksum is a remainder of the sum of each field, from the Start code to Flag/Error, divided by 256 (0x100).
- End code: 1 byte. Indicates the end of a packet. It always should be 0x0A. It is also used as a code indicating the end of binary data such as fingerprint templates.

Function Prototype

```
UF_RET_CODE UF_SendPacket( BYTE command, UINT32 param, UINT32 size,  
BYTE flag, int timeout );
```

Parameters

command

Command field of a packet. Refer to the *Packet Protocol Manual* for available commands.

param

Param field of a packet.

size

Size field of a packet.

flag

Flag field of a packet.

timeout

Sets the timeout in milliseconds. If sending does not complete within this limit, UF_ERR_WRITE_SERIAL_TIMEOUT will be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To send ES command with user ID 10 and ADD_NEW(0x79) option,  
UF_RET_CODE result = UF_SendPacket( UF_COM_ES, 10, 0, 0x79, 2000 );
```

```
If( result != UF_RET_SUCCESS )  
{  
    Return result;  
}
```

3.4.2 UF_SendNetworkPacket

Sends a 15 byte network packet to the specified module. In order to support RS422 or RS485 network interfaces, SFM modules support Network Packet Protocol. Network packet is composed of 15 bytes, whose start code is different from the standard packet, and includes 2 bytes for terminal ID. The terminal ID is equal to the lower 2 bytes of Module ID of system parameter.

Field	Start code	Terminal ID	Command	Param	Size	Flag / Error	Checksum	End code
Bytes	1	2	1	4	4	1	1	1
Value	0x41	1 ~ 0xFFFF	Same as standard protocol				Checksum of 13 bytes	0x0A

Function Prototype

```
UF_RET_CODE UF_SendNetworkPacket( BYTE command, USHORT terminalID,
UINT32 param, UINT32 size, BYTE flag, int timeout );
```

Parameters

command

Command field of a packet. Refer to the *Packet Protocol Manual* for available commands.

terminalID

Specifies the terminal ID of the receiving module.

param

Param field of a packet.

size

Size field of a packet.

flag

Flag field of a packet.

timeout

Sets the timeout in milliseconds. If sending does not complete within this limit, UF_ERR_WRITE_SERIAL_TIMEOUT will be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4.3 UF_ReceivePacket

Receives a 13 byte packet from the module. Most commands of Packet Protocol can be implemented by a pair of UF_SendPacket / UF_ReceivePacket or UF_SendNetworkPacket / UF_ReceiveNetworkPacket.

Function Prototype

```
UF_RET_CODE UF_ReceivePacket( BYTE* packet, int timeout );
```

Parameters

packet

Pointer to the 13 byte packet.

timeout

Sets the timeout in milliseconds. If receiving does not complete within this limit, UF_ERR_READ_SERIAL_TIMEOUT will be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4.4 UF_ReceiveNetworkPacket

Receives a 15 byte network packet from the specified module.

Function Prototype

```
UF_RET_CODE UF_ReceiveNetworkPacket( BYTE* packet, int timeout );
```

Parameters

packet

Pointer to the 15 byte packet.

timeout

Sets the timeout in milliseconds. If receiving does not complete within this limit, UF_ERR_READ_SERIAL_TIMEOUT will be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4.5 UF_SendRawData

Some commands such as ET(Enroll Template) and IT(Identify Template) send additional data after the 13/15 byte request packet. UF_SendRawData is used in these cases for sending the data.

Function Prototype

```
UF_RET_CODE UF_SendRawData( BYTE* buf, UINT32 size, int timeout );
```

Parameters

buf

Pointer to a data buffer.

size

Number of bytes to be sent.

timeout

Sets the timeout in milliseconds. If sending does not complete within this limit, UF_ERR_WRITE_SERIAL_TIMEOUT will be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4.6 UF_ReceiveRawData

Some commands such as ST(Scan Template) and RT(Read Template) return additional data after the 13/15 byte response packet. UF_ReceiveRawData is used in these cases for receiving the data.

Function Prototype

```
UF_RET_CODE UF_ReceiveRawData( BYTE* buf, UINT32 size, int timeout,  
BOOL checkEndCode );
```

Parameters

buf

Pointer to a data buffer.

size

Number of bytes to be received.

timeout

Sets the timeout in milliseconds. If receiving does not complete within this limit, UF_ERR_READ_SERIAL_TIMEOUT will be returned.

checkEndCode

Data transfer ends with '0x0a'. If this parameter is FALSE, the function returns without checking the end code.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4.7 UF_SendDataPacket

Sends data using Extended Data Transfer Protocol. Dividing large data into small blocks can reduce communication errors between the host and the module. Extended Data Transfer Protocol is an extension of Packet Protocol to provide a reliable and customizable communication for large data. In Extended Data Transfer Protocol, data is divided into multiple data packets. And a data packet consists of fixed-length header, variable-length data body, and 4 byte checksum. Commands which use the Extended Data Transfer Protocols are EIX, VIX, IIX, RIX, SIX, and UG.

Function Prototype

```
UF_RET_CODE UF_SendDataPacket( BYTE command, BYTE* buf, UINT32
dataSize, UINT32 dataPacketSize );
```

Parameters

command

Command field of a packet. Valid commands are EIX, VIX, IIX, RIX, SIX, and UG.

buf

Pointer to a data buffer.

dataSize

Number of bytes to be sent.

dataPacketSize

Size of data packet. For example, if dataSize is 16384 bytes and dataPacketSize is 4096 bytes, the data will be divided into 4 data packets.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4.8 UF_ReceiveDataPacket

Receives data using Extended Data Transfer Protocol. The size of data packet should be specified before calling this function.

Function Prototype

```
UF_RET_CODE UF_ReceiveDataPacket( BYTE command, BYTE* buf, UINT32  
dataSize );
```

Parameters

command

Command field of a packet. Valid commands are EIX, VIX, IIX, RIX, SIX, and UG.

buf

Pointer to a data buffer.

dataSize

Number of bytes to be received.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.4.9 UF_SetSendPacketCallback

If SendPacketCallback is specified, it is called after sending a packet successfully. The argument of the callback is the pointer to the packet.

Function Prototype

```
void UF_SetSendPacketCallback( void (*callback) ( BYTE* ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

Example

See CMainFrame::SendPacketCallback in UniFingerUI source codes.

3.4.10 UF_SetReceivePacketCallback

If ReceivePacketCallback is specified, it is called after receiving a packet successfully. The argument of the callback is the pointer to the received packet.

Function Prototype

```
void UF_SetReceivePacketCallback( void (*callback)( BYTE* ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

Example

See CMainFrame::ReceivePacketCallback in UniFingerUI source codes.

3.4.11 UF_SetSendDataPacketCallback

If `SendDataPacketCallback` is specified, it is called after sending a data packet successfully. The argument of the callback is the index of the data packet and the number of total data packets.

Function Prototype

```
void UF_SetSendDataPacketCallback( void (*callback)( int index, int  
numOfPacket ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

Example

See `CMainFrame::DataPacketCallback` in `UniFingerUI` source codes.

3.4.12 UF_SetReceiveDataPacketCallback

If ReceiveDataPacketCallback is specified, it is called after receiving a data packet successfully. The argument of the callback is the index of the data packet and the number of total data packets.

Function Prototype

```
void UF_SetReceiveDataPacketCallback( void (*callback)( int index,  
int numOfPacket ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

Example

See CMainFrame::DataPacketCallback in UniFingerUI source codes.

3.4.13 UF_SetSendRawDataCallback

If `SendRawDataCallback` is specified, it is called during sending raw data. The argument of the callback is the written length and the total length of data.

Function Prototype

```
void UF_SetSendRawDataCallback( void (*callback)( int writtenLen, int  
totalSize ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

Example

See `CMainFrame::RawDataCallback` in `UniFingerUI` source codes.

3.4.14 UF_SetReceiveRawDataCallback

If ReceiveRawDataCallback is specified, it is called during receiving data. The argument of the callback is the read length and the total length of data.

Function Prototype

```
void UF_SetReceiveRawDataCallback( void (*callback)( int readLen, int  
totalSize ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

Example

See CMainFrame::RawDataCallback in UniFingerUI source codes.

3.4.15 UF_SetDefaultPacketSize

Sets the size of data packets used in Extended Data Transfer protocol. The default value is 4096. When BEACon is used as an Ethernet-to-Serial converter, this value should not be larger than 256.

Function Prototype

```
void UF_SetDefaultPacketSize( int defaultSize );
```

Parameters

defaultSize

Size of data packet.

Return Values

None

3.4.16 UF_GetDefaultPacketSize

Returns the size of data packet used in Extended Data Transfer protocol.

Function Prototype

```
int UF_GetDefaultPacketSize();
```

Parameters

None

Return Values

The size of data packet.

3.5 Generic Command API

The commands defined in the *Packet Protocol Manual* can be classified into several categories according to the types of packet exchange. Generic Command API provides functions which encapsulate these categories. Like low-level Packet API, users need not to call these functions directly. Most commands have corresponding high-level API in the SDK.

- UF_Command: encapsulates the commands composed of one request packet and one response packet.
- UF_CommandEx: encapsulates the commands composed of one request packet and multiple response packets.
- UF_CommandSendData: encapsulates the commands which send additional data after a request packet.
- UF_CommandSendDataEx: encapsulates the commands which send additional data and have multiple response packets.
- UF_Cancel: cancels the previously issued command.
- UF_SetProtocol: sets the type of packet protocol.
- UF_GetProtocol: gets the type of packet protocol.
- UF_GetModuleID: gets the module ID.
- UF_SetGenericCommandTimeout: sets the timeout for generic commands.
- UF_SetInputCommandTimeout: sets the timeout for commands which require user inputs.
- UF_GetGenericCommandTimeout: gets the timeout for generic commands.
- UF_GetInputCommandTimeout: gets the timeout for commands which require user inputs.
- UF_SetNetworkDelay: sets the delay for the Network Packet Protocol.
- UF_GetNetworkDelay: gets the delay for the Network Packet Protocol.

3.5.1 UF_Command

Encapsulates the commands composed of one request packet and one response packet. The majority of commands can be implemented using UF_Command.

Function Prototype

```
UF_RET_CODE UF_Command( BYTE command, UINT32* param, UINT32* size,  
BYTE* flag );
```

Parameters

command

Command field of a packet. Refer to the *Packet Protocol Manual* for available commands.

param

Param field of a packet. This parameter is used both for input and output.

size

Size field of a packet. This parameter is used both for input and output.

flag

Flag field of a packet. This parameter is used both for input and output.

Return Values

If packets are transferred successfully, return UF_RET_SUCCESS. Otherwise, return the corresponding error code. UF_RET_SUCCESS only means that request packet is received successfully. To know if the operation succeeds, the flag field should be checked.

Example

```
// To read Timeout(0x62) system parameter,  
UINT32 param = 0;
```

```

UINT32 size = 0;
BYTE flag = 0x62;
UINT32 timeout;

UF_RET_CODE result = UF_Command( UF_COM_SR, &param, &size, &flag );

If( result != UF_RET_SUCCESS ) // communication error
{
    return result;
}

If( flag != UF_PROTO_RET_SUCCESS ) // protocol error
{
    return UF_GetErrorCode( result );
}

// succeed
timeout = size;

```

3.5.2 UF_CommandEx

Encapsulates the commands composed of one request packet and multiple response packets. Command such as ES(Enroll) and IS(Identify) can have more than one response packet. To handle these cases, UF_CommandEx requires a message callback function, which should return TRUE when the received packet is the last one.

Function Prototype

```
UF_RET_CODE UF_CommandEx( BYTE command, UINT32* param, UINT32* size,
BYTE* flag, BOOL (*msgCallback)(BYTE) );
```

Parameters

command

Command field of a packet. Refer to the *Packet Protocol Manual* for available commands.

param

Param field of a packet. This parameter is used both for input and output.

size

Size field of a packet. This parameter is used both for input and output.

flag

Flag field of a packet. This parameter is used both for input and output.

msgCallback

Pointer to the callback function. This callback is called when a response packet is received. If the callback return TRUE, UF_CommandEx will return immediately. If the callback return FALSE, UF_CommandEx will wait for another response packet.

Return Values

If packets are transferred successfully, return UF_RET_SUCCESS. Otherwise, return the corresponding error code. UF_RET_SUCCESS only means that request packet is

received successfully. To know if the operation succeeds, the flag field should be checked.

Example

```
// UF_Identify() is implemented as follows;
/**
 *      Message callback for identification
 */
BOOL UF_IdentifyMsgCallback( BYTE errCode )
{
    if( errCode == UF_PROTO_RET_SCAN_SUCCESS )
    {
        if( s_IdentifyCallback )
        {
            (*s_IdentifyCallback)( errCode );
        }

        return FALSE;
    }
    else
    {
        return TRUE;
    }
}

/**
 *      Identify
 */
UF_RET_CODE UF_Identify( UINT32* userID, BYTE* subID )
{
    UINT32 param = 0;
    UINT32 size = 0;
    BYTE flag = 0;

    int result = UF_CommandEx( UF_COM_IS, &param, &size, &flag,
UF_IdentifyMsgCallback );
}
```

```
    if( result != UF_RET_SUCCESS )
    {
        return result;
    }
    else if( flag != UF_PROTO_RET_SUCCESS )
    {
        return UF_GetErrorCode( flag );
    }

    *userID = param;
    *subID = size;

    return UF_RET_SUCCESS;
}
```


3.5.3 UF_CommandSendData

Encapsulates the commands which send additional data after a request packet. For example, GW(Write GPIO Configuration) command should send configuration data after the request packet.

Function Prototype

```
UF_RET_CODE UF_CommandSendData( BYTE command, UINT32* param, UINT32*  
size, BYTE* flag, BYTE* data, UINT32 dataSize );
```

Parameters

command

Command field of a packet. Refer to the *Packet Protocol Manual* for available commands.

param

Param field of a packet. This parameter is used both for input and output.

size

Size field of a packet. This parameter is used both for input and output.

flag

Flag field of a packet. This parameter is used both for input and output.

data

Pointer to the data buffer to be sent.

dataSize

Number of bytes to be sent.

Return Values

If packets are transferred successfully, return UF_RET_SUCCESS. Otherwise, return the corresponding error code. UF_RET_SUCCESS only means that request packet is received successfully. To know if the operation succeeds, the flag field should be

checked.

3.5.4 UF_CommandSendDataEx

Encapsulates the commands which send additional data and have multiple response packets. For example, ET(Enroll Template) command sends template data after request packet and can have multiple response packets.

Function Prototype

```
UF_RET_CODE UF_CommandSendDataEx( BYTE command, UINT32* param,
UINT32* size, BYTE* flag, BYTE* data, UINT32 dataSize, BOOL
(*msgCallback)(BYTE), BOOL waitUserInput );
```

Parameters

command

Command field of a packet. Refer to the *Packet Protocol Manual* for available commands.

param

Param field of a packet. This parameter is used both for input and output.

size

Size field of a packet. This parameter is used both for input and output.

flag

Flag field of a packet. This parameter is used both for input and output.

data

Pointer to the data buffer to be sent.

dataSize

Number of bytes to be sent.

msgCallback

Pointer to the callback function. This callback is called when a response packet is received. If the callback return TRUE, UF_CommandSendDataEx will return immediately. If the callback return FALSE, UF_CommandSendDataEx will wait

for another response packet.

waitUserInput

TRUE if the command needs user input. Otherwise, FALSE.

Return Values

If packets are transferred successfully, return UF_RET_SUCCESS. Otherwise, return the corresponding error code. UF_RET_SUCCESS only means that request packet is received successfully. To know if the operation succeeds, the flag field should be checked.

3.5.5 UF_Cancel

Cancels the command which is being processed by the module. When the module is executing a command which needs user input to proceed, the status of the module will be changed to UF_SYS_BUSY. If users want to execute another command before finishing the current one, they can explicitly cancel it by this function.

Function Prototype

```
UF_RET_CODE UF_Cancel( BOOL receivePacket );
```

Parameters

receivePacket

If TRUE, UF_Cancel waits until the response packet is received. If FALSE, UF_Cancel returns immediately after sending the request packet.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.5.6 UF_SetProtocol

Selects packet protocol. If the host connects to the single module through RS232 interface, use UF_SINGLE_PROTOCOL. If there are multiple modules in RS422/485 networks, use UF_NETWORK_PROTOCOL. The protocol should also be compatible with the Network Mode system parameter.

Network Mode	Supported Protocol	
	13 byte Packet Protocol	15 byte Network Packet Protocol
Single(0x30)	O	O
Network (0x31/0x32)	X	O

Function Prototype

```
void UF_SetProtocol( UF_PROTOCOL protocol, UINT32 moduleID );
```

Parameters

protocol

UF_SINGLE_PROTOCOL for 13 byte packet protocol, UF_NETWORK_PROTOCOL for 15 byte network packet protocol.

moduleID

Specifies the ID of the module. This parameter is applicable when the protocol is set to UF_NETWORK_PROTOCOL.

Return Values

None

3.5.7 UF_GetProtocol

Gets the selected protocol.

Function Prototype

```
UF_PROTOCOL UF_GetProtocol();
```

Parameters

None

Return Values

UF_SINGLE_PROTOCOL or UF_NETWORK_PROTOCOL.

3.5.8 UF_GetModuleID

Gets the ID of the module.

Function Prototype

```
UINT32 UF_GetModuleID();
```

Parameters

None

Return Values

ID of the module.

3.5.9 UF_SetGenericCommandTimeout

Sets the timeout for generic commands. The default timeout is 2,000ms.

Function Prototype

```
void UF_SetGenericCommandTimeout( int timeout );
```

Parameters

timeout

Specifies the timeout period in milliseconds.

Return Values

None

3.5.10 UF_SetInputCommandTimeout

Sets the timeout for commands which need user input. The default timeout is 10,000ms.

Function Prototype

```
void UF_SetInputCommandTimeout( int timeout );
```

Parameters

timeout

Specifies the timeout period in milliseconds.

Return Values

None

3.5.11 UF_GetGenericCommandTimeout

Gets the timeout for generic commands.

Function Prototype

```
int UF_GetGenericCommandTimeout();
```

Parameters

None

Return Values

Timeout for generic commands.

3.5.12 UF_GetInputCommandTimeout

Gets the timeout for commands which need user input.

Function Prototype

```
int UF_GetInputCommandTimeout();
```

Parameters

None

Return Values

Timeout for commands which need user input.

3.5.13 UF_SetNetworkDelay

In half duplex mode, the same communication lines are shared for sending and receiving data. To prevent packet collisions on the shared line, there should be some delay between receiving and sending data. The default delay is set to 40ms. This value can be optimized for specific environments.

Function Prototype

```
void UF_SetNetworkDelay( int delay );
```

Parameters

delay

Specified the delay in milliseconds.

Return Values

None

3.5.14 UF_GetNetworkDelay

Gets the network delay.

Function Prototype

```
int UF_GetNetworkDelay();
```

Parameters

None

Return Values

Delay in milliseconds.

3.6 Module API

These functions provide basic information about the module.

- UF_GetModuleInfo: gets the basic module information.
- UF_GetModuleString: gets a string describing the module.
- UF_SearchModule: searches a module and find out communication parameters.
- UF_SearchModuleID: searches an ID of a module.
- UF_SearchModuleBySocket: searches a module through socket interface.
- UF_SearchModuleIDEx: searches multiple models in a network.
- UF_CalibrateSensor: calibrates a sensor.
- UF_Upgrade: upgrades firmware.
- UF_Reset: resets the module.
- UF_Lock: locks the module.
- UF_Unlock: unlocks the module.
- UF_ChangePassword: changes the master password of a module.
- UF_PowerOff : Programmatically turns off a module (SFM4000 only.)

3.6.1 UF_GetModuleInfo

Retrieves the type, version and sensor information of the module.

Function Prototype

```
UF_RET_CODE      UF_GetModuleInfo(      UF_MODULE_TYPE*      type,  
UF_MODULE_VERSION* version, UF_MODULE_SENSOR* sensorType );
```

Parameters

type

Available types are as follows;

Value	Description
UF_MODULE_3000	SFM 3000 series modules
UF_MODULE_3500	SFM 3500 series modules
UF_MODULE_4000	SFM 4000 series modules
UF_MODULE_5000	SFM 5000 series modules
UF_MODULE_5500	SFM 5500 series modules
UF_BIOENTRY_SMART	BioEntry Smart
UF_BIOENTRY_PASS	BioEntry Pass

version

Version number of the module.

sensorType

Sensor type of the module.

Value	Sensor Type
UF_SENSOR_FL	Authentec AF-S2
UF_SENSOR_FC	Atmel FingerChip
UF_SENSOR_OP	Optical Sensor I (OP2/OP3/OP4)
UF_SENSOR_TC	UPEK TouchChip (TC1/TC2/TC2S)
UF_SENSOR_OC2	Optical Sensor II (OC2/OD)

UF_SENSOR_TS	UPEK TouchStrip (TS4)
UF_SENSOR_OL	Optical Sensor III (OL)
UF_SENSOR_OH	Optical Sensor IV (OH)

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.6.2 UF_GetModuleString

Retrieves a string that describes the module information. This function should be called after UF_GetModuleInfo.

Function Prototype

```
char* UF_GetModuleString( UF_MODULE_TYPE type, UF_MODULE_VERSION  
version, UF_MODULE_SENSOR sensorType );
```

Parameters

type

Specifies the type of the module.

version

Specifies the version number of the module.

sensorType

Specifies the sensor type of the module.

Return Values

Null-terminated string that describes the module information. This pointer is a static data in the SDK. So, it should not be shared or freed by applications.

3.6.3 UF_SearchModule

Search a module connected to the specified serial port. UF_SearchModule tries all combinations of communication parameters. If it finds any module on the serial port, it returns the communication parameters and its module ID.

Function Prototype

```
UF_RET_CODE UF_SearchModule( const char* port, int* baudrate, BOOL*
asciiMode, UF_PROTOCOL* protocol, UINT32* moduleID, void
(*callback)( const char* comPort, int baudrate ) );
```

Parameters

port

Serial port.

baudrate

Pointer to the baud rate to be returned.

asciiMode

Pointer to the packet translation mode to be returned.

protocol

Pointer to the protocol type to be returned.

moduleID

Pointer to the module ID to be returned.

callback

Pointer to the callback function. The callback function can be used for displaying the progress of the search. This parameter can be NULL.

Return Values

If it finds a module, return UF_RET_SUCCESS. If the search fails, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.6.4 UF_SearchModuleID

Until firmware V1.3, SFM modules respond both standard and network packets regardless of Network Mode system parameter. However, since firmware V1.4, the modules only respond to 15 byte network packets if Network Mode system parameter is not Single(0x30). So, if users don't know ID of the module, they cannot communicate with it in network environments. UF_SerachModuleID can be used to retrieve the ID of the module in these cases. Refer to ID command section in the Packet Protocol Manual for details.

Function Prototype

```
UF_RET_CODE UF_SearchModuleID( UINT32* moduleID );
```

Parameters

moduleID

Pointer to the module ID to be returned.

Return Values

If it finds a module, return UF_RET_SUCCESS. If the search fails, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.6.5 UF_SearchModuleBySocket

Search a module connected to the specified IP address. If it finds any module, it will return the communication parameters and the module ID.

Function Prototype

```
UF_RET_CODE  UF_SearchModuleBySocket(  const  char*  inetAddr,  int
tcpPort,  BOOL*  asciiMode,  UF_PROTOCOL*  protocol,  UINT32*  moduleID );
```

Parameters

inetAddr

IP address.

tcpPort

TCP port.

asciiMode

Pointer to the packet translation mode to be returned.

protocol

Pointer to the protocol type to be returned.

moduleID

Pointer to the module ID to be returned.

Return Values

If it finds a module, return UF_RET_SUCCESS. If the search fails, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.6.6 UF_SearchModuleIDEx

UF_SearchModuleID is used for searching a module. To search multiple modules in a RS422/485 network, UF_SearchModuleIDEx should be used instead. By calling this function repetitively, users can search all the modules connected to a network.

Function Prototype

```
UF_RET_CODE UF_SearchModuleIDEx( unsigned short* foundModuleID, int  
numOfFoundID, unsigned short* moduleID, int* numOfID );
```

Parameters

foundModuleID

Pointer to the array of module IDs, which are already found. When the ID of a module is in this array, the module will ignore the search command.

numOfFoundID

Number of module IDs, which are already found.

moduleID

Pointer to the array of module IDs, which will be filled with newly found IDs.

numOfID

Pointer to the number of module IDs to be returned.

Return Values

If it finds one or more modules, return UF_RET_SUCCESS. If the search fails, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

Example

```
int numOfModuleID;  
unsigned short moduleID[32];
```

```

int numOfFoundID = 0;
BOOL foundNewID = FALSE;

do {
    result = UF_SearchModuleIDEx( moduleID, numOfFoundID, moduleID
+ numOfFoundID, &numOfModuleID );

    if( result == UF_RET_SUCCESS )
    {
        foundNewID = TRUE;

        numOfFoundID += numOfModuleID;
    }
    else
    {
        foundNewID = FALSE;
    }
} while( foundNewID && numOfFoundID < 32 );

```

3.6.7 UF_CalibrateSensor

Calibrates fingerprint sensor. This function is supported for AuthenTec's FingerLoc AF-S2 and UPEK's TouchChip. After using the UF_CalibrateSensor, UF_Save should be called to save calibration data into flash memory.

Function Prototype

```
UF_RET_CODE UF_CalibrateSensor();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.6.8 UF_Upgrade

Upgrades the firmware of the module. Users should not turn off the module when upgrade is in progress.

Function Prototype

```
UF_RET_CODE UF_Upgrade( const char* firmwareFilename, int dataPacketSize );
```

Parameters

firmwareFilename

Null-terminated string that specifies the firmware file name.

dataPacketSize

The packet size of firmware data. If it is 16384, the firmware is divided into 16384 byte packets before transferring to the module.

Return Values

If upgrade succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

With SendDataPacketCallback

3.6.9 UF_Reset

Resets the module.

Function Prototype

```
UF_RET_CODE UF_Reset();
```

Parameters

None

Return Values

UF_RET_SUCCESS

3.6.10 UF_Lock

Locks the module. When the module is locked, it returns UF_ERR_LOCKED to functions other than UF_Unlock.

Function Prototype

```
UF_RET_CODE UF_Lock();
```

Parameters

None

Return Values

If the module is locked successfully, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.6.11 UF_Unlock

Unlocks a locked module.

Function Prototype

```
UF_RET_CODE UF_Unlock( const unsigned char* password );
```

Parameters

password

16 byte master password. The default password is a string of 16 NULL characters.

Return Values

If the password is wrong, return UF_ERR_NOT_MATCH. If it is successful, return UF_RET_SUCCESS.

3.6.12 UF_ChangePassword

Changes the master password. The master password is not the encryption key but the key to lock or unlock the module.

Function Prototype

```
UF_RET_CODE UF_ChangePassword( const unsigned char* newPassword,  
const unsigned char* oldPassword );
```

Parameters

newPassword

16 byte new password.

oldPassword

16 byte old password.

Return Values

If the old password is wrong, return UF_ERR_NOT_MATCH. If it is successful, return UF_RET_SUCCESS.

If you want to use the master password permanently, you have to call UF_Save for saving master password on your module.

3.6.13 UF_PowerOff

Programmatically turns off a module. This function is only available with SFM4000 series.

Function Prototype

```
UF_RET_CODE UF_PowerOff();
```

Parameters

None

Return Values

The module is powered off successfully, return UF_RET_SUCCESS.

3.7 System Parameters API

Functions for managing system parameters. Available system parameters are defined in `UF_SysParameter.h`. See the *Packet Protocol Manual* for available values for each parameter.

This API also provides functions for saving and loading system configurations.

- `UF_InitSysParameter`: clears the system parameter cache.
- `UF_GetSysParameter`: gets the value of a system parameter.
- `UF_SetSysParameter`: sets the value of a system parameter.
- `UF_GetMultiSysParameter`: gets the values of multiple system parameters.
- `UF_SetMultiSysParameter`: sets the values of multiple system parameters.
- `UF_Save`: saves system parameters into the flash memory.
- `UF_SaveConfiguration`: saves system configurations into the specified file.
- `UF_ReadConfigurationHeader`: reads configuration information stored in a file.
- `UF_LoadConfiguration`: loads system configurations from the specified file.
- `UF_MakeParameterConfiguration`: makes parameter configuration data to be saved.

3.7.1 UF_InitSysParameter

To prevent redundant communication, the SFM SDK caches the system parameters previously read or written. UF_InitSysParameter clears this cache. It is called in UF_Reconnect.

Function Prototype

```
void UF_InitSysParameter();
```

Parameters

None

Return Values

None

3.7.2 UF_GetSysParameter

Reads the value of a system parameter.

Function Prototype

```
UF_RET_CODE  UF_GetSysParameter(  UF_SYS_PARAM  parameter,  UINT32*  
value );
```

Parameters

parameter

System parameter to be read.

value

Pointer to the value of the specified system parameter to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. If there is no such parameter, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.7.3 UF_SetSysParameter

Writes the value of a system parameter. The parameter value is changed in memory only. To make the change permanent, UF_Save should be called after this function. For BioEntry Smart and Pass, users cannot change the UF_SYS_MODULE_ID system parameter.

Function Prototype

```
UF_RET_CODE  UF_SetSysParameter(  UF_SYS_PARAM  parameter,  UINT32
value );
```

Parameters

parameter

System parameter to be written.

value

Value of the system parameter. Refer to the *Packet Protocol Manual* for available values for each parameter.

Return Values

If the function succeeds, return UF_RET_SUCCESS. If there is no such parameter, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.7.4 UF_GetMultiSysParameter

Reads the values of multiple system parameters.

Function Prototype

```
UF_RET_CODE      UF_GetMultiSysParameter(      int      parameterCount,  
UF_SYS_PARAM* parameters, UINT32* values );
```

Parameters

parameterCount

Number of system parameters to be read.

parameters

Array of system parameters to be read.

values

Array of the values of the specified system parameters to be read.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To read 3 system parameters, UF_SYS_TIMEOUT, UF_SYS_ENROLL_MODE,  
// UF_SYS_SECURITY_LEVEL,  
  
UF_SYS_PARAM parameters[3] = { UF_SYS_TIMEOUT, UF_SYS_ENROLL_MODE,  
UF_SYS_SECURITY_LEVEL };  
UINT32 values[3];  
  
UF_RET_CODE result = UF_GetMultiSysParameter( 3, parameters, values );
```

3.7.5 UF_SetMultiSysParameter

Writes the values of multiple system parameters. The parameter value is changed in memory only. To make the change permanent, UF_Save should be called.

Function Prototype

```
UF_RET_CODE      UF_SetMultiSysParameter(      int      parameterCount,  
UF_SYS_PARAM* parameters, UINT32* values );
```

Parameters

parameterCount

Number of system parameters to be written.

parameters

Array of system parameters to be written.

values

Array of the values of the specified system parameters to be written.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.7.6 UF_Save

Saves the system parameters into the flash memory.

Function Prototype

```
UF_RET_CODE UF_Save();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.7.7 UF_SaveConfiguration

Saves system configurations into the specified file. The configuration file consists of a file header and multiple configuration components. There are 5 configuration components to be saved.

```
typedef enum {
    UF_CONFIG_PARAMETERS      = 0x01, // System parameters
    UF_CONFIG_GPIO            = 0x02, // GPIO configurations for
                                   // SFM3000
    UF_CONFIG_IO              = 0x04, // IO configurations for SFM3500
                                   // SFM3500
    UF_CONFIG_WIEGAND         = 0x08, // Extended Wiegand
    UF_CONFIG_USER_MEMORY     = 0x10, // User memory
} UF_CONFIG_TYPE;
```

Function Prototype

```
UF_RET_CODE UF_SaveConfiguration( const char* filename, const char*
description,      int      numOfComponent,      UFConfigComponentHeader*
componentHeader, void** componentData );
```

Parameters

filename

Null-terminated string that specifies the file name.

description

Null-terminated string describing the configuration file. The maximum length of description is 256 bytes.

numOfComponent

Number of components to be saved.

componentHeader

Pointer to an array of UFConfigComponentHeader structures to be saved.

```
typedef struct {
    UF_CONFIG_TYPE type;
    UINT32 dataSize;
    UINT32 checksum;
} UFConfigComponentHeader;
```

componentData

Pointer to an array of component data to be saved.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To save system parameters and IO configuration
// of a SFM 3500 module into "SFM3500.cfg" file,
UF_ConfigComponentHeader configHeader[2];
void* configData[2];

// Make system parameter component
UFConfigParameter* parameter = (UFConfigParameter*)malloc( sizeof(int)
+ NUM_OF_PARAMETER * sizeof(UFConfigParameterItem) );
UF_MakeParameterConfiguration( &configHeader[0], (BYTE*)parameter );
configData[0] = (void*)parameter;

// Make IO component
UFConfigIO* io = (UFConfigIO*)malloc( sizeof(UFConfigIO) +
sizeof(UFConfigOutputItem) * (UF_MAX_OUTPUT_EVENT - 1) );
UF_MakeIOConfiguration( &configHeader[1], (BYTE*)io );
configData[1] = (void*)io;

UF_RET_CODE result = UF_SaveConfiguration( "SFM3500.cfg",
"Configuration file for SFM3500", 2, configHeader, configData );
```

3.7.8 UF_ReadConfigurationHeader

Reads the header information from a file which is saved by UF_SaveConfiguration.

Function Prototype

```
UF_RET_CODE    UF_ReadConfigurationHeader(    const    char*    filename,
UFConfigFileHeader* header );
```

Parameters

filename

Null-terminated string that specifies the file name.

header

Pointer to the UFConfigFileHeader to be read.

```
typedef struct {
    UINT32        magicNo; // if valid, UF_VALID_CONFIG_FILE
    UINT32        numOfComponent;
    char    description[256];
} UFConfigFileHeader;
```

Return Values

If the header is read successfully, return UF_RET_SUCCESS. If the file is of invalid type, return UF_ERR_INVALID_FILE. Otherwise, return the corresponding error code.

3.7.9 UF_LoadConfiguration

Loads system configurations into a module from the specified file. To make permanent the configuration changes, UF_Save should be called after UF_LoadConfiguration.

Function Prototype

```
UF_RET_CODE  UF_LoadConfiguration(  const  char*  filename,  int
numOfComponent, UF_CONFIG_TYPE* type );
```

Parameters

filename

Null-terminated string that specifies the file name.

numOfComponent

Number of configuration components to be loaded.

type

Array of component types to be loaded.

Return Values

If the configurations are loaded successfully, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To load system parameters and IO configuration
// of a SFM 3500 module from "SFM3500.cfg" file,
UF_CONFIG_TYPE configType[2] = { UF_CONFIG_PARAMETERS, UF_CONFIG_IO };

UF_RET_CODE  result  =  UF_LoadConfiguration(  "SFM3500.cfg",  2,
configType );
```

3.7.10 UF_MakeParameterConfiguration

Make a UFConfigComponentHeader and a UFConfigParameter structure to be used in UF_SaveConfiguration.

Function Prototype

```
UF_RET_CODE  UF_MakeParameterConfiguration(  UFConfigComponentHeader*  
configHeader, BYTE* configData );
```

Parameters

configHeader

Pointer to the UFConfigComponentHeader structure.

configData

Pointer to the UFConfigParameter structure. It should be preallocated large enough to store all the parameter information.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8 Template Management API

These functions provide template management interfaces such as read, delete and fix. Users can also manage user ids and administration levels associated with templates using these APIs.

- `UF_GetNumOfTemplate`: gets the number of template stored in a module.
- `UF_GetMaxNumOfTemplate`: gets the template capacity of a module.
- `UF_GetAllUserInfo`: gets all the template and user information stored in a module.
- `UF_GetAllUserInfoEx`: gets all the template and user information stored in a BioEntry reader.
- `UF_SortUserInfo`: sorts `UFUserInfo` structures.
- `UF_SetUserInfoCallback`: sets the callback function for getting user information.
- `UF_SetAdminLevel`: sets the administration level of a user.
- `UF_GetAdminLevel`: gets the administration level of a user.
- `UF_ClearAllAdminLevel`: clears all the administration levels of users.
- `UF_SaveDB`: saves templates and user information into the specified file.
- `UF_LoadDB`: loads templates and user information from the specified file.
- `UF_CheckTemplate`: checks if the specified ID has templates.
- `UF_ReadTemplate`: reads the templates of the specified user ID.
- `UF_ReadOneTemplate`: reads one template of the specified user ID.
- `UF_SetScanCallback`: sets the callback function for scanning fingerprints.
- `UF_ScanTemplate`: scans a fingerprint on the sensor and retrieves the fingerprint template.
- `UF_FixProvisionalTemplate`: saves the provisional templates into the flash memory.
- `UF_SetSecurityLevel`: sets the security level of a user.

- UF_GetSecurityLevel: gets the security level of a user.
- UF_SetAuthType: sets the authentication type of a user.
- UF_GetAuthType: gets the authentication type of a user.
- UF_GetUserIDByAuthType: gets the user IDs with specified authentication type.
- UF_ResetAllAuthType: resets the authentication types of all users.
- UF_SetEntranceLimit: sets the entrance limit of a user.
- UF_GetEntranceLimit: gets the entrance limit of a user.
- UF_ClearAllEntranceLimit: clears the entrance limits of all users.

3.8.1 UF_GetNumOfTemplate

Gets the number of templates stored in the module.

Function Prototype

```
UF_RET_CODE UF_GetNumOfTemplate( UINT32* numOfTemplate );
```

Parameters

numOfTemplate

Pointer to the number of templates to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.2 UF_GetMaxNumOfTemplate

Gets the template capacity of the module.

Function Prototype

```
UF_RET_CODE UF_GetMaxNumOfTemplate( UINT32* maxNumOfTemplate );
```

Parameters

maxNumOfTemplate

Pointer to the template capacity to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.3 UF_GetAllUserInfo

Retrieves all the user and template information stored in the module.

Function Prototype

```
UF_RET_CODE    UF_GetAllUserInfo(    UFUserInfo*    userInfo,    UINT32*
numOfUser,    UINT32* numOfTemplate );
```

Parameters

userInfo

Array of UFUserInfo structures, which will store all the information. This pointer should be preallocated large enough to store all the information. UFUserInfo structure is defined as follows;

```
typedef struct {
    UINT32      userID;
    BYTE    numOfTemplate;
    BYTE    adminLevel; // See UF_SetAdminLevel
    BYTE    securityLevel; // See UF_SetSecurityLevel
    BYTE    reserved;
} UFUserInfo;
```

numOfUser

Pointer to the number of users to be returned.

numOfTemplate

Pointer to the number of templates to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
UINT32 maxUser;
UINT32 numOfUser, numOfTemplate;

UF_RET_CODE result = UF_GetSysParameter( UF_SYS_ENROLLED_FINGER,
&maxUser );

UFUserInfo* userInfo = (UFUserInfo*)malloc( maxUser *
sizeof(UFUserInfo) );

result = UF_GetAllUserInfo( userInfo, &numOfUser, &numOfTemplate );
```


3.8.4 UF_GetAllUserInfoEx

Retrieves all the user and template information stored in the BioEntry reader.

Function Prototype

```
UF_RET_CODE  UF_GetAllUserInfoEx(  UFUserInfoEx*  userInfo,  UINT32*
numOfUser,  UINT32*  numOfTemplate );
```

Parameters

userInfo

Array of UFUserInfoEx structures, which will store all the information. This pointer should be preallocated large enough to store all the information. UFUserInfoEx structure is defined as follows;

```
typedef struct {
    UINT32      userID;
    UINT32      checksum[10]; // checksum of each template data
    BYTE  numOfTemplate;
    BYTE  adminLevel;
    BYTE  duress[10];
    BYTE  securityLevel;
} UFUserInfoEx;
```

numOfUser

Pointer to the number of users to be returned.

numOfTemplate

Pointer to the number of templates to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.5 UF_SortUserInfo

Sorts an UFUserInfo array in ascending order of user ID.

Function Prototype

```
void UF_SortUserInfo( UFUserInfo* userInfo, int numOfWork );
```

Parameters

userInfo

Array of UFUserInfo structures.

numOfWork

Number of UFUserInfo.

Return Values

None

3.8.6 UF_SetUserInfoCallback

Sets the callback function for getting user information. It is also called when enrolling templates in **UF_LoadDB** and reading templates in **UF_SaveDB**.

Function Prototype

```
void UF_SetUserInfoCallback( void (*callback)( int index, int  
numOfTemplate ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

Example

See CMainFrame::UserInfoCallback in UniFingerUI source codes.

3.8.7 UF_SetAdminLevel

Sets the administration level of a user. See `UF_EnrollAfterVerification` and `UF_DeleteAllAfterVerification` for usage of administration level.

Function Prototype

```
UF_RET_CODE    UF_SetAdminLevel(    UINT32    userID,    UF_ADMIN_LEVEL
adminLevel );
```

Parameters

userID

User ID.

adminLevel

Specifies the administration level of the user.

Value	Note
UF_ADMIN_NONE	
UF_ADMIN_ENROLL	Can enroll users.
UF_ADMIN_DELETE	Can delete users.
UF_ADMIN_ALL	Can enroll and delete users.

Return Values

If the function succeeds, return `UF_RET_SUCCESS`. Otherwise, return the corresponding error code.

3.8.8 UF_GetAdminLevel

Gets the administration level of a user.

```
UF_RET_CODE  UF_GetAdminLevel(  UINT32  userID,  UF_ADMIN_LEVEL*  
adminLevel  )
```

Parameters

userID

User ID.

adminLevel

Pointer to the administration level of the user to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.9 UF_ClearAllAdminLevel

Resets administration levels of all users to UF_ADMIN_NONE.

Function Prototype

```
UF_RET_CODE UF_ClearAllAdminLevel();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.10 UF_SaveDB

Saves all the templates and user information stored in a module into the specified file.

Function Prototype

```
UF_RET_CODE UF_SaveDB( const char* fileName );
```

Parameters

fileName

Null-terminated string that specifies the file name.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.11 UF_LoadDB

Loads templates and user information from the specified file. All the templates previously stored in the module will be erased before loading the DB.

Function Prototype

```
UF_RET_CODE UF_LoadDB( const char* fileName );
```

Parameters

fileName

Null-terminated string that specifies the file name.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.12 UF_CheckTemplate

Checks if the specified user ID has enrolled templates.

Function Prototype

```
UF_RET_CODE UF_CheckTemplate( UINT32 userID, UINT32* numOfTemplate );
```

Parameters

userID

User ID.

numOfTemplate

Pointer to the number of templates of the user ID to be returned.

Return Values

If there are templates of the user ID, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.13 UF_ReadTemplate

Reads the templates of the specified user ID.

Function Prototype

```
UF_RET_CODE UF_ReadTemplate( UINT32 userID, UINT32* numOfTemplate,  
BYTE* templateData );
```

Parameters

userID

User ID.

numOfTemplate

Pointer to the number of templates of the user ID to be returned.

templateData

Pointer to the template data to be returned. This pointer should be preallocated large enough to store all the template data.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.14 UF_ReadOneTemplate

Reads one template of the specified user ID.

Function Prototype

```
UF_RET_CODE UF_ReadOneTemplate( UINT32 userID, int subID, BYTE*  
templateData );
```

Parameters

userID

User ID.

subID

Sub index of the template. It is between 0 and 9.

templateData

Pointer to the template data to be returned. This pointer should be preallocated large enough to store all the template data.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.15 UF_SetScanCallback

Sets the callback function of scanning fingerprints. This callback is called when SCAN_SUCCESS message is received.

Function Prototype

```
void UF_SetScanCallback( void (*callback) ( BYTE ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

3.8.16 UF_ScanTemplate

Scans a fingerprint on the sensor and receives the template of it.

Function Prototype

```
UF_RET_CODE UF_ScanTemplate( BYTE* templateData, UINT32* templateSize,  
UINT32* imageQuality );
```

Parameters

templateData

Pointer to the template data to be returned.

templateSize

Pointer to the template size to be returned.

imageQuality

Pointer to the image quality score to be returned. The score shows the quality of scanned fingerprint and is in the range of 0 ~ 100.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.17 UF_FixProvisionalTemplate

UF_SYS_PROVISIONAL_ENROLL determines if enrolled templates are saved permanently into flash memory or temporarily into DRAM. With provisional enroll, enrolled templates on DRAM will be erased if the module is turned off. UF_FixProvisionalTemplate saves the provisional templates into the flash memory.

Function Prototype

```
UF_RET_CODE UF_FixProvisionalTemplate();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.18 UF_SetSecurityLevel

Since V1.6 firmware, the security level can be assigned per user basis for 1:1 matching. 1:N matching – identification – is not affected by this setting.

Function Prototype

```
UF_RET_CODE      UF_SetSecurityLevel(      UINT32      userID,  
UF_USER_SECURITY_LEVEL securityLevel );
```

Parameters

userID

User ID.

securityLevel

Specifies the security level of the user.

Value

UF_USER_SECURITY_DEFAULT

UF_USER_SECURITY_1_TO_1000

...

UF_USER_SECURITY_1_TO_100000000

Note

Same as defined by Security Level system parameter

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.19 UF_GetSecurityLevel

Gets the security level of a user.

Function Prototype

```
UF_RET_CODE UF_GetSecurityLevel( UINT32 userID, UF_SECURITY_LEVEL*  
securityLevel );
```

Parameters

userID

User ID.

securityLevel

Pointer to the security level of the user to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.20 UF_SetAuthType

Sets the authentication type of a user. UF_AUTH_BYPASS can be used for 1:1 matching, when it is necessary to allow access without matching fingerprints. UF_AUTH_REJECT can be used for disabling some IDs temporarily. The default authentication mode is UF_AUTH_FINGERPRINT.

Function Prototype

```
UF_RET_CODE UF_SetAuthType( UINT32 userID, UF_AUTH_TYPE authType );
```

Parameters

userID

User ID.

authType

Specifies the authentication type of the user.

Value	Note
UF_AUTH_FINGERPRINT	Fingerprint authentication.
UF_AUTH_BYPASS	Authentication will succeed without matching fingerprints.
UF_AUTH_REJECT	Authentication will always fail.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.21 UF_GetAuthType

Gets the authentication type of a user.

Function Prototype

```
UF_RET_CODE UF_GetAuthType( UINT32 userID, UF_AUTH_TYPE* authType );
```

Parameters

userID

User ID.

authType

Pointer to the authentication type of the user to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.22 UF_GetUserIDByAuthType

Receive user IDs with the specified authentication type.

Function Prototype

```
UF_RET_CODE  UF_GetUserIDByAuthType(  UF_AUTH_TYPE  authType,  int*  
numOfID, UINT32* userID );
```

Parameters

authType

Authentication type.

numOfID

Pointer to the number of user IDs to be returned.

userID

Array of user IDs which have the specified authentication type.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.23 UF_ResetAllAuthType

Resets the authentication types of all users to UF_AUTH_FINGERPRINT.

Function Prototype

```
UF_RET_CODE UF_ResetAllAuthType();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.24 UF_SetEntranceLimit

Specifies how many times the user is permitted to access per day. The available options are between 0 and 7. The default value is 0, which means that there is no limit. If the user tries to authenticate after the limit is reached, UF_ERR_EXCEED_ENTRANCE_LIMIT error will be returned.

Function Prototype

```
UF_RET_CODE UF_SetEntranceLimit( UINT32 userID, int entranceLimit );
```

Parameters

userID

User ID.

entranceLimit

Entrance limit between 0 and 7.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.25 UF_GetEntranceLimit

Gets the entrance limit of a user.

Function Prototype

```
UF_RET_CODE UF_GetEntranceLimit( UINT32 userID, int* entranceLimit,  
int* entranceCount );
```

Parameters

userID

User ID.

entranceLimit

Pointer to the entrance limit of the user.

entranceCount

Pointer to the number of entrance for today. This count is reset to 0 at midnight.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.8.26 UF_ClearAllEntranceLimit

Resets the entrance limits of all users to 0 – infinite.

Function Prototype

```
UF_RET_CODE UF_ClearAllEntranceLimit();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.9 Image Manipulation API

UFIImage is a data structure for exchanging image data between the host and the module. It consists of 28 byte header and raw image data.

```
typedef struct {  
    int    width; // width of the fingerprint image  
    int    height; // height of the fingerprint image  
    int    compressed; // compression status - currently not used  
    int    encrypted; // encryption status - currently not used  
    int    format; // 0- gray, 1- binary, 2- 4bit gray  
    int    imgLen; // width * height  
    int    templateLen; // size of fingerprint template  
  
    // currently not used  
    BYTE buffer[1]; // pointer to the raw pixel data  
} UFIImage;
```

- UF_ConvertToBitmap: converts a UFIImage structure into HBITMAP.
- UF_SaveImage: saves a UFIImage structure into BMP file.
- UF_LoadImage: loads a BMP file and convert it into a UFIImage structure.
- UF_ReadImage: retrieves the last scanned fingerprint image.
- UF_ScanImage: scans a fingerprint on the sensor and retrieves the image data.

3.9.1 UF_ConvertToBitmap

To display a UImage on the PC screen, it should be converted to a bitmap first. UF_ConvertToBitmap converts a UImage into a device independent bitmap and returns the handle of it. After drawing the bitmap, it should be destroyed by calling DeleteObject().

Function Prototype

```
HBITMAP UF_ConvertToBitmap( UImage* image );
```

Parameters

image

Pointer to the UImage structure to be converted.

Return Values

If the function succeeds, return the HBITMAP of the bitmap. Otherwise, return NULL.

Example

```
// The following snippet is from FingerprintViewer.cpp in UniFingerUI
// source codes
class CFingerprintViewer : public CStatic
{
    // ...
private:
    UImage* m_Image;
    HBITMAP m_Bitmap;
};

void CFingerprintViewer::OnPaint()
{
    CPaintDC dc( this );
    CBitmap bmp;
```

```

        if( m_Bitmap )
        {
            DeleteObject( m_Bitmap );
        }

        if( m_Image )
        {
            m_Bitmap = UF_ConvertToBitmap( m_Image );
            bmp.Attach( m_Bitmap );
        }
        else
        {
            bmp.LoadBitmap( IDB_LOGO );
        }

        CDC bmDC;
        bmDC.CreateCompatibleDC( &dc );
        CBitmap *pOldbmp = bmDC.SelectObject( &bmp );

        BITMAP bi;
        bmp.GetBitmap( &bi );

        CRect rect;
        this->GetClientRect( &rect );

        dc.SetStretchBltMode( HALFTONE );
        dc.StretchBlt( 1, 1, rect.Width() - 2, rect.Height() - 2, &bmDC,
0, 0, bi.bmWidth, bi.bmHeight, SRCCOPY );

        bmDC.SelectObject( pOldbmp );
    }

```

3.9.2 UF_SaveImage

Converts a UFIImage into a bitmap and save it into the specified file.

Function Prototype

```
UF_RET_CODE UF_SaveImage( const char* fileName, UFIImage* image );
```

Parameters

fileName

Null-terminated string that specifies the file name.

image

Pointer to the UFIImage to be saved.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.9.3 UF_LoadImage

Loads a bmp file into a UImage structure.

Function Prototype

```
UF_RET_CODE UF_LoadImage( const char* fileName, UImage* image );
```

Parameters

fileName

Null-terminated string that specifies the file name.

image

Pointer to the UImage structure.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.9.4 UF_ReadImage

Reads the last scanned fingerprint image.

Function Prototype

```
UF_RET_CODE UF_ReadImage( UFIImage* image );
```

Parameters

image

Pointer to the UFIImage structure.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
UFIImage* image = (UFIImage*)malloc( UF_IMAGE_HEADER_SIZE +  
UF_MAX_IMAGE_SIZE );  
UF_RET_CODE result = UF_ReadImage( image );
```

3.9.5 UF_ScanImage

Scans a fingerprint input on the sensor and retrieves the image of it.

Function Prototype

```
UF_RET_CODE UF_ScanImage( UFIImage* image );
```

Parameters

image

Pointer to the UFIImage structure.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.10 Enroll API

There are three ways to enroll fingerprints; scanning live fingerprints, using scanned images, or sending templates acquired elsewhere. The enrollment process varies according to `UF_SYS_ENROLL_MODE` parameter. Users can also fine tune the enrollment process by selecting enroll options.

- `UF_Enroll`: enrolls fingerprint inputs on the sensor.
- `UF_EnrollContinue`: continues the enrollment process when the enroll mode is `UF_ENROLL_TWO_TIMES2` or `UF_ENROLL_TWO_TEMPLATES2`.
- `UF_EnrollAfterVerification`: enrolls after an administrator is verified.
- `UF_EnrollTemplate`: enrolls a template.
- `UF_EnrollMultipleTemplates`: enrolls multiple templates to the specified ID.
- `UF_EnrollImage`: enrolls a fingerprint image.
- `UF_SetEnrollCallback`: sets the callback function for enrollment process.

3.10.1 UF_Enroll

Enrolls fingerprint inputs on the sensor. The enrollment process varies according to the UF_SYS_ENROLL_MODE system parameter.

Enroll Mode	Description
UF_ENROLL_ONE_TIME	Scans a fingerprint and enrolls it.
UF_ENROLL_TWO_TIMES1	Scans two fingerprints and enrolls the better one of the two. The scanning of the second fingerprint starts automatically.
UF_ENROLL_TWO_TIMES2	Same as UF_ENROLL_TWO_TIMES1, but the scanning of the second fingerprint should be initiated by another request packet.
UF_ENROLL_TWO_TEMPLATES1	Scans two fingerprints and enrolls both of them. The scanning of the second fingerprint starts automatically.
UF_ENROLL_TWO_TEMPLATES2	Same as UF_ENROLL_TWO_TEMPLATES1, but the scanning of the second fingerprint should be initiated by another request packet.

Since the SFM SDK v3.3.0, Additional enumerations relevant to the UF_ENROLL_MODE have been added for the clarifying naming rules. Also, it's compatible as below.

```
UF_ENROLL_MODE_ONE_TIME = UF_ENROLL_ONE_TIME,  
UF_ENROLL_MODE_TWO_TIMES1 = UF_ENROLL_TWO_TIMES1,  
UF_ENROLL_MODE_TWO_TIMES2 = UF_ENROLL_TWO_TIMES2,  
UF_ENROLL_MODE_TWO_TEMPLATES1 = UF_ENROLL_TWO_TEMPLATES1,  
UF_ENROLL_MODE_TWO_TEMPLATES2 = UF_ENROLL_TWO_TEMPLATES2,
```

Users can also fine tune the enrollment process by selecting one of the following

UF_ENROLL_OPTIONS.

Option	Description
UF_ENROLL_NONE	Overwrites existing templates of the same ID.
UF_ENROLL_ADD_NEW	Adds templates to the same user ID. The maximum number of templates per user is 10.
UF_ENROLL_AUTO_ID	The user ID will be assigned automatically by the module.
UF_ENROLL_CHECK_ID	Before enrolling, checks if the user ID has already some templates. If it does, UF_ERR_EXIST_ID will be returned. This option is useful when users do not want to overwrite existing templates.
UF_ENROLL_CHECK_FINGER	Before enrolling, checks if the same fingerprint is already enrolled. If the identification succeeds, return UF_ERR_EXIST_FINGER error. If the identification fails, continue enroll process with UF_ENROLL_ADD_NEW option.
UF_ENROLL_CHECK_FINGER_AUTO_ID	Before enrolling, checks if the same fingerprint is already enrolled. If the identification succeeds, return UF_ERR_EXIST_FINGER error. If the identification fails, continue enroll process with UF_ENROLL_AUTO_ID option.
UF_ENROLL_DURESS	Adds another fingerprint as duress finger to the specified user ID. Under duress, users can authenticate with duress finger to notify the threat. When duress finger is matched, the module will return UF_ERR_DURESS_FINGER error code and write a log. Users can also setup output signals for duress events. When enrolling, the duress finger should not match with non-duress fingerprints of the same ID. If it is the case, UF_ERR_EXIST_FINGER error will be returned.

Since the SFM SDK v3.3.0, Additional enumerations relevant to the UF_ENROLL_OPTIONS have been added for the clarifying naming rules. Also, it's compatible as below.

```
UF_ENROLL_OPTION_NONE = UF_ENROLL_NONE,  
UF_ENROLL_OPTION_ADD_NEW = UF_ENROLL_ADD_NEW,  
UF_ENROLL_OPTION_AUTO_ID = UF_ENROLL_AUTO_ID,  
UF_ENROLL_OPTION_CONTINUE = UF_ENROLL_CONTINUE,  
UF_ENROLL_OPTION_CHECK_ID = UF_ENROLL_CHECK_ID,  
UF_ENROLL_OPTION_CHECK_FINGER = UF_ENROLL_CHECK_FINGER,  
UF_ENROLL_OPTION_CHECK_FINGER_AUTO_ID = UF_ENROLL_CHECK_FINGER_AUTO_ID,  
UF_ENROLL_OPTION_DURESS = UF_ENROLL_DURESS,
```

Function Prototype

```
UF_RET_CODE  UF_Enroll(  UINT32  userID,  UF_ENROLL_OPTION  option,  
UINT32* enrollID, UINT32* imageQuality );
```

Parameters

userID

User ID.

option

Enroll option.

enrollID

Pointer to the enrolled user ID. This parameter can be different from userID when AUTO_ID option is used.

imageQuality

Pointer to the image quality score to be returned. The score shows the quality of scanned fingerprint and is in the range of 0 ~ 100.

Return Values

If enroll succeeds, return `UF_RET_SUCCESS`. Otherwise, return the corresponding error code.

3.10.2 UF_EnrollContinue

Continues the enrollment process when the enroll mode is UF_ENROLL_TWO_TIMES2 or UF_ENROLL_TWO_TEMPLATES2.

Function Prototype

```
UF_RET_CODE UF_EnrollContinue(    UINT32  userID,  UINT32*  enrollID,  
UINT32*  imageQuality );
```

Parameters

userID

User ID.

enrollID

Pointer to the enrolled user ID. This parameter can be different from userID when AUTO_ID option is used.

imageQuality

Pointer to the image quality score to be returned. The score shows the quality of scanned fingerprint and is in the range of 0 ~ 100.

Return Values

If enroll succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To enroll user ID 10 with enroll option of ADD_NEW,  
UINT32 mode;  
UF_RET_CODE result = UF_GetSysParameter( UF_SYS_ENROLL_MODE, &mode );  
  
UINT32 userID, imageQuality;  
result = UF_Enroll( 10, UF_ENROLL_ADD_NEW, &userID, &imageQuality );
```

```
// If enroll mode is UF_ENROLL_TWO_TIMES2 or UF_ENROLL_TWO_TEMPLATES2
If( result == UF_RET_SUCCESS
    && (mode == UF_ENROLL_TWO_TEMPLATES2
        || mode == UF_ENROLL_TWO_TIMES2 ) )
{
    result = UF_EnrollContinue( 10, &userID, &imageQuality );
}
```

3.10.3 UF_EnrollAfterVerification

Enroll and Delete functions change the fingerprint DB stored in the module. For some applications, it might be necessary to obtain administrator's permission before enrolling or deleting fingerprints. To process these functions, a user with proper administration level should verify himself first. If there is no user with corresponding administration level, these commands will fail with UF_ERR_UNSUPPORTED error code. If the verification fails, UF_ERR_NOT_MATCH error code will be returned. The only exception is that UF_EnrollAfterVerification will succeed when the fingerprint DB is empty. In that case, the first user enrolled by UF_EnrollAfterVerification will have UF_ADMIN_LEVEL_ALL.

Function Prototype

```
UF_RET_CODE      UF_EnrollAfterVerification(      UINT32      userID,  
UF_ENROLL_OPTION option, UINT32* enrollID, UINT32* imageQuality );
```

Parameters

userID

User ID.

option

Enroll option.

enrollID

Pointer to the enrolled user ID. This parameter can be different from userID when AUTO_ID option is used.

imageQuality

Pointer to the image quality score to be returned. The score shows the quality of scanned fingerprint and is in the range of 0 ~ 100.

Return Values

If enroll succeeds, return UF_RET_SUCCESS. If there is no user with corresponding

administration level, return `UF_ERR_UNSUPPORTED`. If administrator's verification fails, return `UF_ERR_NOT_MATCH`. Otherwise, return the corresponding error code.

3.10.4 UF_EnrollTemplate

Enrolls a fingerprint template.

Function Prototype

```
UF_RET_CODE UF_EnrollTemplate( UINT32 userID, UF_ENROLL_OPTION option,  
UINT32 templateSize, BYTE* templateData, UINT32* enrollID );
```

Parameters

userID

User ID.

option

Enroll option.

templateSize

Size of the template data.

templateData

Pointer to the template data.

enrollID

Pointer to the enrolled user ID. This parameter can be different from userID when AUTO_ID option is used.

Return Values

If enroll succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.10.5 UF_EnrollMultipleTemplates

Enrolls multiple templates to the specified ID.

Function Prototype

```
UF_RET_CODE      UF_EnrollMultipleTemplates(      UINT32      userID,  
UF_ENROLL_OPTION option, int numOfTemplate,  UINT32  templateSize,  
BYTE* templateData, UINT32* enrollID );
```

Parameters

userID

User ID.

option

Enroll option.

numOfTemplate

Number of templates to be enrolled.

templateSize

Size of one template data. For example, when enroll 3 templates of 384 byte, this parameter is 384 not 1152.

templateData

Pointer to the template data.

enrollID

Pointer to the enrolled user ID. This parameter can be different from userID when AUTO_ID option is used.

Return Values

If enroll succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.10.6 UF_EnrollImage

Enrolls a fingerprint image.

Function Prototype

```
UF_RET_CODE UF_EnrollImage( UINT32 userID, UF_ENROLL_OPTION option,  
UINT32 imageSize, BYTE* imageData, UINT32* enrollID, UINT32*  
imageQuality );
```

Parameters

userID

User ID.

option

Enroll option.

imageSize

Size of the image data.

imageData

Pointer to the raw image data. Note that it is not the pointer to UFIImage, but the pointer to the raw pixel data without the UFIImage header.

enrollID

Pointer to the enrolled user ID. This parameter can be different from userID when AUTO_ID option is used..

imageQuality

Pointer to the image quality score to be returned. The score shows the quality of scanned fingerprint and is in the range of 0 ~ 100.

Return Values

If enroll succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.10.7 UF_SetEnrollCallback

Sets the callback function for enrollment process. This callback is called after receiving response packets with UF_PROTO_RET_SCAN_SUCCESS, UF_PROTO_RET_SUCCESS, or UF_PROTO_RET_CONTINUE messages.

Function Prototype

```
void UF_SetEnrollCallback( void (*callback)( BYTE errCode,  
UF_ENROLL_MODE enrollMode, int numOfSuccess ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

3.11 Identify API

Checks if a fingerprint input is among the enrolled user ids. While verification checks only the fingerprints of a specified user id, identification searches all the enrolled fingerprints until a match is found. As in enrollment, there are three ways to identify fingerprints; scanning live fingerprints, using scanned images, or sending templates acquired elsewhere.

- `UF_Identify`: identifies the fingerprint input on the sensor.
- `UF_IdentifyTemplate`: identifies a template.
- `UF_IdentifyImage`: identifies a fingerprint image.
- `UF_SetIdentifyCallback`: sets the callback function for identification.

3.11.1 UF_Identify

Identifies the fingerprint input on the sensor.

Function Prototype

```
UF_RET_CODE UF_Identify( UINT32* userID, BYTE* subID );
```

Parameters

userID

Pointer to the user ID to be returned.

subID

Pointer to the index of the template to be returned.

Return Values

If matching succeeds, return UF_RET_SUCCESS. If matching fails, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.11.2 UF_IdentifyTemplate

Identifies a template.

Function Prototype

```
UF_RET_CODE    UF_IdentifyTemplate(    UINT32    templateSize,    BYTE*  
templateData, UINT32* userID, BYTE* subID );
```

Parameters

templateSize

Size of the template data.

templateData

Pointer to the template data.

userID

Pointer to the user ID to be returned.

subID

Pointer to the index of the template to be returned.

Return Values

If matching succeeds, return UF_RET_SUCCESS. If matching fails, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.11.3 UF_IdentifyImage

Identifies a fingerprint image.

Function Prototype

```
UF_RET_CODE UF_IdentifyImage(  UINT32  imageSize,  BYTE*  imageData,  
UINT32* userID, BYTE* subID );
```

Parameters

imageSize

Size of the image data.

imageData

Pointer to the raw image data. Note that it is not the pointer to UFIImage, but the pointer to the raw pixel data without the UFIImage header.

userID

Pointer to the user ID to be returned.

subID

Pointer to the index of the template to be returned.

Return Values

If matching succeeds, return UF_RET_SUCCESS. If matching fails, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.11.4 UF_SetIdentifyCallback

Sets the callback function for identification process. This callback is called after receiving UF_PROTO_RET_SCAN_SUCCESS message.

Function Prototype

```
void UF_SetIdentifyCallback( void (*callback) ( BYTE ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

3.12 Verify API

Verifies if a fingerprint input matches the enrolled fingerprints of the specified user id. As in enroll process, there are three ways to verify fingerprints; scanning live fingerprints, using scanned images, or sending templates acquired elsewhere.

- `UF_Verify`: verifies the fingerprint input on the sensor.
- `UF_VerifyTemplate`: verifies a template.
- `UF_VerifyHostTemplate`: verifies the fingerprint input on the sensor with the templates sent by the host.
- `UF_VerifyImage`: verifies a fingerprint image.
- `UF_SetVerifyCallback`: sets the callback function for verification process.

3.12.1 UF_Verify

Verifies if a fingerprint input on the sensor matches the enrolled fingerprints of the specified user id.

Function Prototype

```
UF_RET_CODE UF_Verify( UINT32 userID, BYTE* subID );
```

Parameters

userID

User ID.

subID

Pointer to the index of the template to be returned.

Return Values

If matching succeeds, return UF_RET_SUCCESS. If matching fails, return UF_ERR_NOT_MATCH. Otherwise, return the corresponding error code.

3.12.2 UF_VerifyTemplate

Verifies a template.

Function Prototype

```
UF_RET_CODE    UF_VerifyTemplate(    UINT32    templateSize,    BYTE*  
templateData, UINT32 userID, BYTE* subID );
```

Parameters

templateSize

Size of the template data.

templateData

Pointer to the template data to be sent.

userID

User ID.

subID

Pointer to the index of the template to be returned.

Return Values

If matching succeeds, return UF_RET_SUCCESS. If matching fails, return UF_ERR_NOT_MATCH. Otherwise, return the corresponding error code.

3.12.3 UF_VerifyHostTemplate

Transmits fingerprint templates from the host to the module and verifies if they match the live fingerprint input on the sensor.

Function Prototype

```
UF_RET_CODE  UF_VerifyHostTemplate(  UINT32  numOfTemplate,  UINT32  
templateSize, BYTE* templateData );
```

Parameters

numOfTemplate

Number of templates to be transferred to the module.

templateSize

Size of a template.

templateData

Pointer to the template data to be transferred to the module.

Return Values

If matching succeeds, return UF_RET_SUCCESS. If matching fails, return UF_ERR_NOT_MATCH. Otherwise, return the corresponding error code.

3.12.4 UF_VerifyImage

Verifies a fingerprint image.

Function Prototype

```
UF_RET_CODE UF_VerifyImage( UINT32 imageSize, BYTE* imageData, UINT32  
userID, BYTE* subID );
```

Parameters

imageSize

Size of the fingerprint image.

imageData

Pointer to the raw image data. Note that it is not the pointer to UFIImage, but the pointer to the raw pixel data without the UFIImage header.

userID

User ID.

subID

Pointer to the index of the template to be returned.

Return Values

If matching succeeds, return UF_RET_SUCCESS. If matching fails, return UF_ERR_NOT_MATCH. Otherwise, return the corresponding error code.

3.12.5 UF_SetVerifyCallback

Sets the callback function for verification process. The callback function is called after receiving UF_PROTO_RET_SCAN_SUCCESS message.

Function Prototype

```
void UF_SetVerifyCallback( void (*callback) ( BYTE ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

3.13 Delete API

Provides functions for deleting stored templates.

- `UF_Delete`: deletes the templates of the specified user ID.
- `UF_DeleteOneTemplate`: deletes one template of the specified user ID.
- `UF_DeleteMultipleTemplates`: deletes the template of multiple user IDs.
- `UF_DeleteAll`: deletes all the templates.
- `UF_DeleteAllAfterVerification`: deletes templates after administrator's verification.
- `UF_SetDeleteCallback`: sets the callback function for delete process.

3.13.1 UF_Delete

Deletes the enrolled templates of the specified user ID.

Function Prototype

```
UF_RET_CODE UF_Delete( UINT32 userID );
```

Parameters

userID

User ID.

Return Values

If delete succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.13.2 UF_DeleteOneTemplate

Deletes one template of the specified user ID.

Function Prototype

```
UF_RET_CODE UF_DeleteOneTemplate( UINT32 userID, int subID );
```

Parameters

userID

User ID.

subID

Sub index of the template. It is between 0 and 9.

Return Values

If delete succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.13.3 UF_DeleteMultipleTemplates

Deletes the enrolled templates of multiple user IDs.

Function Prototype

```
UF_RET_CODE UF_DeleteMultipleTemplates( UINT32 startUserID, UINT32
lastUserID, int* deletedUserID );
```

Parameters

startUserID

First user ID to be deleted.

lastUserID

Last user ID to be deleted.

deletedUserID

Pointer to the number of IDs to be actually deleted by the module.

Return Values

If delete succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// Delete templates of ID 10 ~ ID 20
int numOfDeleted;
UF_RET_CODE result = UF_DeleteMultipleTemplates( 10, 20,
&numOfDeleted );
```

3.13.4 UF_DeleteAll

Deletes all the templates stored in a module.

Function Prototype

```
UF_RET_CODE UF_DeleteAll();
```

Parameters

None

Return Values

If delete succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.13.5 UF_DeleteAllAfterVerification

Deletes all the templates after administrator's verification.

Function Prototype

```
UF_RET_CODE UF_DeleteAllAfterVerification();
```

Parameters

None

Return Values

If delete succeeds, return UF_RET_SUCCESS. If there is no user with corresponding administration level, return UF_ERR_UNSUPPORTED. If administrator's verification fails, return UF_ERR_NOT_MATCH. Otherwise, return the corresponding error code.

3.13.6 UF_SetDeleteCallback

Sets the callback function for delete process. This callback is called after receiving UF_PROTO_RET_SCAN_SUCCESS or UF_PROTO_RET_CONTINUE.

Function Prototype

```
void UF_SetDeleteCallback( void (*callback) ( BYTE ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

None

3.14 IO API for SFM3500

SFM3500 modules have three input ports, three output ports, and three LED ports which are configurable for specific functions. For BioEntry Smart and Pass, there are two input ports, two output ports, and 2 LED ports. These functions are provided to configure these IO ports.

- UF_InitIO: clears the output event caches.
- UF_SetInputFunction: sets the function of an input port.
- UF_GetInputFunction: gets the function of an input port.
- UF_GetInputStatus: gets the status of an input port.
- UF_GetOutputEventList: gets the output events list assigned to an output port.
- UF_ClearAllOutputEvent: clears all the output events assigned to an output port.
- UF_ClearOutputEvent: clears an output event assigned to an output port.
- UF_SetOutputEvent: adds an output event to an output port.
- UF_GetOutputEvent: gets the signal data of an output event.
- UF_SetOutputStatus: sets the status of an output port.
- UF_SetLegacyWiegandConfig: sets the Wiegand format.
- UF_GetLegacyWiegandConfig: gets the Wiegand format.
- UF_MakeIOConfiguration: makes IO configuration data to be saved into a file.

3.14.1 UF_InitIO

To prevent redundant communication, the SFM SDK caches the output events previously read or written. UF_InitIO clears the cache. It is called in UF_Reconnect.

Function Prototype

```
void UF_InitIO();
```

Parameters

None

Return Values

None

3.14.2 UF_SetInputFunction

Sets the function of an input port. Available functions are as follows;

Function	Description
UF_INPUT_NO_ACTION	No action
UF_INPUT_ENROLL	Enroll
UF_INPUT_IDENTIFY	Identify
UF_INPUT_DELETE	Delete
UF_INPUT_DELETE_ALL	Delete all
UF_INPUT_ENROLL_BY_WIEGAND	Enroll by Wiegand ID
UF_INPUT_VERIFY_BY_WIEGAND	Verify by Wiegand ID
UF_INPUT_DELETE_BY_WIEGAND	Delete by Wiegand ID
UF_INPUT_ENROLL_VERIFICATION	Enroll after administrator's verification
UF_INPUT_ENROLL_BY_WIEGAND _VERIFICATION	Enroll by Wiegand ID after administrator's verification
UF_INPUT_DELETE_VERIFICATION	Delete after administrator's verification
UF_INPUT_DELETE_BY_WIEGAND _VERIFICATION	Delete by Wiegand ID after administrator's verification
UF_INPUT_DELETE_ALL _VERIFICATION	Delete all after administrator's verification
UF_INPUT_CANCEL	Cancel
UF_INPUT_TAMPER_SWITCH_IN	Tamper switch. When the tamper switch is on, Tamper Switch On(0x64) event occurred. When it gets off, Tamper Switch Off(0x65) event occurred. Both events are recorded in log, too. In BioEntry Smart and Pass, UF_INPUT_PORT2 is assigned to Tamper SW.
UF_INPUT_RESET	Reset the module

Function Prototype

```
UF_RET_CODE UF_SetInputFunction( UF_INPUT_PORT port, UF_INPUT_FUNC
inputFunction, UINT32 minimumTime );
```

Parameters

port

One of the three input ports - UF_INPUT_PORT0, UF_INPUT_PORT1, and UF_INPUT_PORT2. For BioEntry Smart and Pass, UF_INPUT_PORT2 is assigned to Tamper SW and not configurable.

inputFunction

Input function.

minimumTime

Minimum duration after which the input signal is acknowledged as active.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.3 UF_GetInputFunction

Gets the function assigned to an input port.

Function Prototype

```
UF_RET_CODE UF_GetInputFunction( UF_INPUT_PORT port, UF_INPUT_FUNC*  
inputFunction, UINT32* minimumTime );
```

Parameters

port

Input port.

inputFunction

Pointer to the input function to be returned.

minimumTime

Pointer to the minimum duration of input signal to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.4 UF_GetInputStatus

Gets the status of an input port.

Function Prototype

```
UF_RET_CODE UF_GetInputStatus( UF_INPUT_PORT port, BOOL remainStatus,  
UINT32* status );
```

Parameters

port

Input port.

remainStatus

If TRUE, don't change the status of the input port after reading. If FALSE, clear the status of the input port.

status

Pointer to the status of the input port to be read. 0 for inactive and 1 for active status.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.5 UF_GetOutputEventList

Gets the list of output events assigned to an output/LED port. Available output events are as follows;

Category	Event
Enroll	UF_OUTPUT_ENROLL_WAIT_WIEGAND
	UF_OUTPUT_ENROLL_WAIT_FINGER
	UF_OUTPUT_ENROLL_PROCESSING
	UF_OUTPUT_ENROLL_BAD_FINGER
	UF_OUTPUT_ENROLL_SUCCESS
	UF_OUTPUT_ENROLL_FAIL
Verify	UF_OUTPUT_VERIFY_WAIT_WIEGAND
	UF_OUTPUT_VERIFY_WAIT_FINGER
	UF_OUTPUT_VERIFY_PROCESSING
	UF_OUTPUT_VERIFY_BAD_FINGER
	UF_OUTPUT_VERIFY_SUCCESS
	UF_OUTPUT_VERIFY_FAIL
Identify	UF_OUTPUT_IDENTIFY_WAIT_FINGER
	UF_OUTPUT_IDENTIFY_PROCESSING
	UF_OUTPUT_IDENTIFY_BAD_FINGER
	UF_OUTPUT_IDENTIFY_SUCCESS
	UF_OUTPUT_IDENTIFY_FAIL
Delete	UF_OUTPUT_DELETE_WAIT_WIEGAND
	UF_OUTPUT_DELETE_WAIT_FINGER
	UF_OUTPUT_DELETE_PROCESSING
	UF_OUTPUT_DELETE_BAD_FINGER
	UF_OUTPUT_DELETE_SUCCESS
	UF_OUTPUT_DELETE_FAIL
Detect	UF_OUTPUT_DETECT_INPUT0
	UF_OUTPUT_DETECT_INPUT1
	UF_OUTPUT_DETECT_INPUT2

	UF_OUTPUT_DETECT_WIEGAND
	UF_OUTPUT_DETECT_FINGER
End Processing	UF_OUTPUT_END_PROCESSING
Duress	UF_OUTPUT_VERIFY_DURESS
	UF_OUTPUT_IDENTIFY_DURESS
Tamper SW	UF_OUTPUT_TAMPER_SWITCH_ON
	UF_OUTPUT_TAMPER_SWITCH_OFF
System	UF_OUTPUT_SYS_STARTED
SmartCard (Available only for BioEntry Smart)	UF_OUTPUT_DETECT_SMARTCARD
	UF_OUTPUT_BAD_SMARTCARD
	UF_OUTPUT_WAIT_SMARTCARD

Function Prototype

```
UF_RET_CODE      UF_GetOutputEventList(      UF_OUTPUT_PORT      port,
UF_OUTPUT_EVENT* events, int* numOfEvent );
```

Parameters

port

Output/LED port.

events

Array of output events to be returned.

numOfEvent

Pointer to the number of output events to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To read the output events assigned to OUT0
```

```
UF_OUTPUT_EVENT events[UF_MAX_OUTPUT_PER_PORT];  
int numOfEvent;  
  
UF_RET_CODE result = UF_GetOutputEventList( UF_OUTPUT_PORT0, events,  
&numOfEvent );
```

3.14.6 UF_ClearAllOutputEvent

Clears all the output events assigned to an output/LED port.

Function Prototype

```
UF_RET_CODE UF_ClearAllOutputEvent( UF_OUTPUT_PORT port );
```

Parameters

port

Output/LED port.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.7 UF_ClearOutputEvent

Clears the specified output event from the output port.

Function Prototype

```
UF_RET_CODE UF_ClearOutputEvent( UF_OUTPUT_PORT port, UF_OUTPUT_EVENT  
event );
```

Parameters

port

Output/LED port.

event

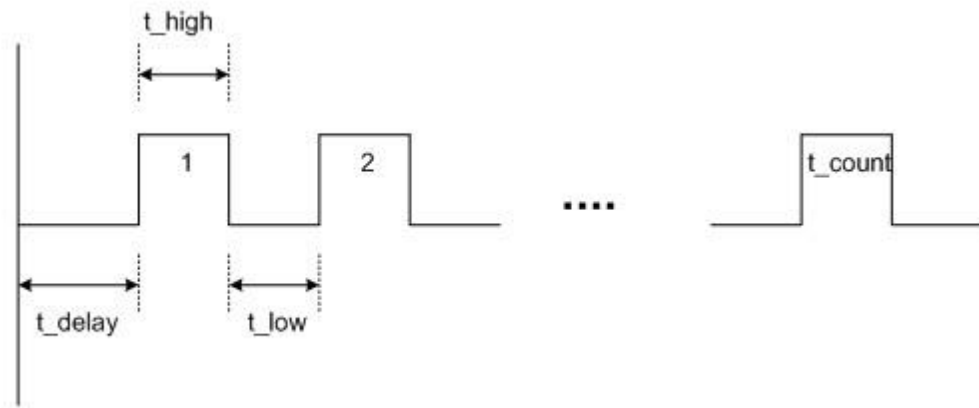
Output event to be deleted from the output/LED port.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.8 UF_SetOutputEvent

Adds an output event to the specified output/LED port. The characteristics of output signal is also specified.



```
typedef struct {  
    unsigned short delay;      // t_delay  
    unsigned short count;     // t_count  
    unsigned short active;    // t_high  
    unsigned short inactive; // t_low  
} UFOutputSignal;
```

Function Prototype

```
UF_RET_CODE UF_SetOutputEvent( UF_OUTPUT_PORT port, UF_OUTPUT_EVENT  
event, UFOutputSignal signal );
```

Parameters

port

Output/LED port.

event

Output event to be added.

signal

Signal specification of the output event.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To turn on the LED0 for 500ms when verification succeeds,
UFOutputSignal signal;
signal.delay = 0;
signal.count = 1;
signal.active = 500;
signal.inactive = 0;

UF_RET_CODE result = UF_SetOutputEvent( UF_OUTPUT_LED0,
UF_OUTPUT_VERIFY_SUCCESS, signal );
```

3.14.9 UF_GetOutputEvent

Gets the signal data of an output event on the specified output/LED port.

Function Prototype

```
UF_RET_CODE UF_GetOutputEvent( UF_OUTPUT_PORT port, UF_OUTPUT_EVENT  
event, UFOutputSignal* signal );
```

Parameters

port

Output/LED port.

event

Output event.

signal

Pointer to the signal data to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.10 UF_SetOutputStatus

Sets the status of an output/LED port.

Function Prototype

```
UF_RET_CODE UF_SetOutputStatus( UF_OUTPUT_PORT port, BOOL status );
```

Parameters

port

Output/LED port.

status

TRUE for active and FALSE for inactive status.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.11 UF_SetLegacyWiegandConfig(Deprecated)

Until the firmware V1.3, SFM 3500 modules only support 26 bit Wiegand format. Since the firmware V1.4, Extended Wiegand Interface is provided, which is much more powerful and flexible. UF_SetLegacyWiegandConfig configures the legacy 26 bit Wiegand format and is provided only for firmwares older than V1.4.

Function Prototype

```
UF_RET_CODE    UF_SetLegacyWiegandConfig(    BOOL    enableInput,    BOOL
enableOutput, UINT32 fcBits, UINT32 fcCode );
```

Parameters

enableInput

If TRUE, the module starts verification process when receiving Wiegand inputs.

enableOutput

If TRUE, the module outputs the user ID when verification or identification succeeds.

fcBits

Specifies the number of facility bits.

fcCode

Specifies the facility code.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.12 UF_GetLegacyWiegandConfig(Deprecated)

Gets the configuration of the 26 bit Wiegand format.

Function Prototype

```
UF_RET_CODE  UF_GetLegacyWiegandConfig(  BOOL*  enableInput,  BOOL*  
enableOutput, UINT32* fcBits, UINT32* fcCode );
```

Parameters

enableInput

Pointer to the input enable status.

enableOutput

Pointer to the output enable status.

fcBits

Pointer to the number of facility bits to be returned.

fcCode

Pointer to the facility code to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.14.13 UF_MakeIOConfiguration

Makes IO configuration data to be saved into a file. See the example of UF_SaveConfiguration for the usage of this function.

Function Prototype

```
UF_RET_CODE      UF_MakeIOConfiguration(      UFConfigComponentHeader*  
configHeader, BYTE* configData );
```

Parameters

configHeader

Pointer to the configuration header to be returned.

configData

Pointer to the configuration data to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15 GPIO API for SFM3000/4000/5000/6000

SFM3000 series modules have 8 GPIO ports, which are configurable for specific functions. The function of each GPIO can be read and programmed via these APIs. . GPIO port can be configured as input, output, shared I/O, Wiegand input, or Wiegand output. In the input mode, enroll, identify, and delete functions are supported. In the output mode, the port can send output patterns corresponding to the various events such as match success, enroll fail, and so on.

- UF_GetGPIOConfiguration: gets the configuration data of a GPIO port.
- UF_SetInputGPIO: configures an input GPIO port.
- UF_SetOutputGPIO: configures an output GPIO port.
- UF_SetSharedGPIO: configures a shared I/O port.
- UF_DisableGPIO: disables a GPIO port.
- UF_ClearAllGPIO: clears all the GPIO configurations.
- UF_SetDefaultGPIO: resets to default GPIO configurations.
- UF_EnableWiegandInput: enables Wiegand input.
- UF_EnableWiegandOutput: enables Wiegand output.
- UF_DisableWiegandInput: disables Wiegand input.
- UF_DisableWiegandOutput: disables Wiegand output.
- UF_MakeGPIOConfiguration: makes GPIO configuration data to be saved into a file.

3.15.1 UF_GetGPIOConfiguration

Gets the configuration data of a GPIO port.

Function Prototype

```
UF_RET_CODE UF_GetGPIOConfiguration( UF_GPIO_PORT port, UF_GPIO_MODE*  
mode, int* numOfData, UFGPIOData* data );
```

Parameters

port

GPIO port from UF_GPIO_0 to UF_GPIO_7.

mode

Pointer to the GPIO mode to be returned. Available GPIO modes are as follows;

Mode	Description
UF_GPIO_INPUT	Input port
UF_GPIO_OUTPUT	Output port
UF_GPIO_SHARED_IO	Shared IO port
UF_GPIO_WIEGAND_INPUT	Wiegand input port
UF_GPIO_WIEGAND_OUTPUT	Wiegand output port

numOfData

Number of configuration data assigned to the port.

data

Array of GPIO configuration data to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To retrieve the configuration data of GPIO 0
UFGPIOData configData[UF_MAX_GPIO_OUTPUT_EVENT];
UF_GPIO_MODE mode;
int numOfData;

UF_RET_CODE result = UF_GetGPIOConfiguration( UF_GPIO_0, &mode,
&numOfData, configData );
```

3.15.2 UF_SetInputGPIO

Configures an input GPIO port.

Function Prototype

```
UF_RET_CODE  UF_SetInputGPIO(  UF_GPIO_PORT  port,  UFGPIOInputData
data );
```

Parameters

port

GPIO port. Only GPIO 0 to GPIO 3 can be an input port.

data

UFGPIOInputData is defined as follows;

```
typedef struct {
    unsigned short  inputFunction; // See UF_GPIO_INPUT_FUNC
                                // in UF_3000IO.h
    unsigned short  activationLevel; // See UF_GPIO_INPUT_ACTIVATION
                                // in UF_3000IO.h
    unsigned short  timeout; // valid only if inputFunction is
                                // UF_GPIO_IN_DELETE_ALL or
                                // UF_GPIO_IN_DELETE_ALL_VERIFICATION
    unsigned short  reserved;
} UFGPIOInputData;
```

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
// To assign identify to GPIO 0 with ACTIVE_HIGH signal
UFGPIOInputData inputData;
```

```
inputData.inputFunction = UF_GPIO_IN_IDENTIFY;  
inputData.activationLevel = UF_GPIO_IN_ACTIVE_HIGH;  
  
UF_RET_CODE result = UF_SetInputGPIO( UF_GPIO_0, inputData );
```

3.15.3 UF_SetOutputGPIO

Configures an output GPIO port.

Function Prototype

```
UF_RET_CODE UF_SetOutputGPIO( UF_GPIO_PORT port, int numOfData,
    UFGPIOOutputData* data );
```

Parameters

port

GPIO port.

numOfData

Number of UFGPIOOutputData to be assigned to the GPIO port.

data

Array of UFGPIOOutputData to be assigned to the GPIO port.

UFGPIOOutputData is defined as follows;

```
typedef struct {
    unsigned short event; // see UF_GPIO_OUTPUT_EVENT
                          // in UF_3000IO.h
    unsigned short level; // see UF_GPIO_OUTPUT_LEVEL
    unsigned short interval; // in milliseconds
    unsigned short blinkingPeriod; // valid only if the level is
                                   // UF_GPIO_OUT_HIGH_BLINK
                                   //or UF_GPIO_OUT_LOW_BLINK
} UFGPIOOutputData;
```

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.4 UF_SetSharedGPIO

Configures the shared I/O GPIO port.

Function Prototype

```
UF_RET_CODE UF_SetSharedGPIO( UF_GPIO_PORT port, UFGPIOInputData  
inputData, int numOfOutputData, UFGPIOOutputData* outputData );
```

Parameters

port

GPIO port. Only GPIO 0 to GPIO 3 can be a shared I/O port.

inputData

Input data to be assigned.

numOfOutputData

Number of UFGPIOOutputData to be assigned.

outputData

Array of UFGPIOOutputData to be assigned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.5 UF_DisableGPIO

Disables a GPIO port.

Function Prototype

```
UF_RET_CODE UF_DisableGPIO( UF_GPIO_PORT port );
```

Parameters

port

GPIO port.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.6 UF_ClearAllGPIO

Clears all the configurations of GPIO ports.

Function Prototype

```
UF_RET_CODE UF_ClearAllGPIO();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.7 UF_SetDefaultGPIO

Resets the configurations of GPIO ports to default.

Function Prototype

```
UF_RET_CODE UF_SetDefaultGPIO();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.8 UF_EnableWiegandInput

Enables Wiegand input. Wiegand input uses GPIO 2 and GPIO 3 as input signal.

Function Prototype

```
UF_RET_CODE UF_EnableWiegandInput( UFGPIOWiegandData data );
```

Parameters

data

UFGPIOWiegandData is defined as follows;

```
typedef struct {  
    unsigned short totalBits;    // fixed at 26 bit  
    unsigned short fcBits;      // number of facility bits  
    unsigned short idBits;      // number of id bits  
    unsigned short fcCode;      // facility code  
} UFGPIOWiegandData;
```

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.9 UF_EnableWiegandOutput

Enables Wiegand output. Wiegand output use GPIO 4 and GPIO 5 as output signal.

Function Prototype

```
UF_RET_CODE UF_EnableWiegandOutput( UFGPIOWiegandData data );
```

Parameters

data

Wiegand configuration data.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.10 UF_DisableWiegandInput

Disables Wiegand input.

Function Prototype

```
UF_RET_CODE UF_DisableWiegandInput();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.11 UF_DisableWiegandOutput

Disables Wiegand output.

Function Prototype

```
UF_RET_CODE UF_DisableWiegandOutput();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.15.12 UF_MakeGPIOConfiguration

Makes GPIO configuration data to be saved into a file.

Function Prototype

```
UF_RET_CODE      UF_MakeGPIOConfiguration(      UFConfigComponentHeader*  
configHeader, BYTE* configData );
```

Parameters

configHeader

Pointer to the configuration header.

configData

Pointer to the configuration data.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

See CUniFingerUI3000IOView::OnGpioSaveFile in UniFingerUI source codes.

3.16 User Memory API

SFM modules reserve 256 bytes for user data. This area can be read and written by these APIs.

- `UF_WriteUserMemory`: writes data to the user memory.
- `UF_ReadUserMemory`: reads the contents of the user memory.

3.16.1 UF_WriteUserMemory

Writes data into the user memory.

Function Prototype

```
UF_RET_CODE UF_WriteUserMemory( BYTE* memory );
```

Parameters

memory

Pointer to the 256 byte array to be written.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.16.2 UF_ReadUserMemory

Reads the contents of the user memory.

Function Prototype

```
UF_RET_CODE UF_ReadUserMemory( BYTE* memory );
```

Parameters

memory

Pointer to the 256 byte array to be read.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17 Log Management API

SFM3500 modules provide logging facility for recording important events. Users can receive logs from the module and delete unwanted ones. The format of a log record is as follows;

Item	Description	Size
Source	UF_LOG_SOURCE_HOST_PORT	1 byte
	UF_LOG_SOURCE_AUX_PORT	
	UF_LOG_SOURCE_WIEGAND_INPUT	
	UF_LOG_SOURCE_IN0	
	UF_LOG_SOURCE_IN1	
	UF_LOG_SOURCE_IN2	
	UF_LOG_SOURCE_FREESCAN	
	UF_LOG_SOURCE_SMARTCARD	
Event ID	One of the UF_OUTPUT_EVENT	1 byte
Date	(DD << 16) (MM << 8) YY	3 bytes
Time	(ss << 16) (mm << 8) hh	3 bytes
User ID	User ID	4 bytes
Custom Field	Customizable by user	4 bytes

In V1.6 firmware, there are two enhancements for logging functions. First, 4 byte custom field is added to log records. Making use of this field, users can add customized events to log records. Second, the log cache is added for real-time monitoring.

- UF_SetTime: sets the time of the module.
- UF_GetTime: gets the time of the module.
- UF_GetNumOfLog: gets the number of log records.
- UF_ReadLog: reads log records.
- UF_ReadLatestLog: reads latest log records.

- UF_DeleteOldestLog: deletes oldest log records.
- UF_DeleteAllLog: deletes all the log records.
- UF_ClearLogCache: clears the log cache.
- UF_ReadLogCache: reads the log records in the cache.
- UF_SetCustomLogField: sets the custom field of log records.
- UF_GetCustomLogField: gets the custom field of log records.

3.17.1 UF_SetTime

Sets the time of the module.

Function Prototype

```
UF_RET_CODE UF_SetTime( time_t timeVal );
```

Parameters

timeVal

Number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.2 UF_GetTime

Gets the time of the module.

Function Prototype

```
UF_RET_CODE UF_GetTime( time_t* timeVal );
```

Parameters

timeVal

Pointer to the time value to be returned by the module.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.3 UF_GetNumOfLog

Retrieves the number of log records.

Function Prototype

```
UF_RET_CODE UF_GetNumOfLog( int* numOfLog, int* numOfTotalLog );
```

Parameters

numOfLog

Pointer to the number of log records to be returned.

numOfTotalLog

Pointer to the maximum log records to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.4 UF_ReadLog

Reads log records.

Function Prototype

```
UF_RET_CODE UF_ReadLog( int startIndex, int count, UFLogRecord*  
logRecord, int* readCount );
```

Parameters

startIndex

Start index of log records to be read.

count

Number of log records to be read.

logRecord

Pointer to the log records to be read.

readCount

Pointer to the number of log records actually read.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.5 UF_ReadLatestLog

Reads latest log records.

Function Prototype

```
UF_RET_CODE UF_ReadLatestLog( int count, UFLogRecord* logRecord, int* readCount );
```

Parameters

count

Number of latest log records to be read.

logRecord

Pointer to the log records to be read.

readCount

Pointer to the number of log records actually read.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.6 UF_DeleteOldestLog

Deletes oldest log records.

Function Prototype

```
UF_RET_CODE UF_DeleteOldestLog( int count, int* deletedCount );
```

Parameters

count

Number of oldest log records to be deleted. It should be a multiple of 256.

deletedCount

Pointer to the number of log records actually deleted.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.7 UF_DeleteAllLog

Deletes all the log records.

Function Prototype

```
UF_RET_CODE UF_DeleteAllLog();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.8 UF_ClearLogCache

Clears the log cache.

Function Prototype

```
UF_RET_CODE UF_ClearLogCache();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.9 UF_ReadLogCache

Reads the log records in the cache. After reading, the cache will be cleared. The maximum number of log records in the cache is 256.

Function Prototype

```
UF_RET_CODE UF_ReadLogCache( int dataPacketSize, int* numOfLog,  
UFLogRecord* logRecord );
```

Parameters

dataPacketSize

Data packet size used in Extended Data Transfer protocol.

numOfLog

Pointer to the number of log records to be returned.

logRecord

Pointer to the log records to be returned. This pointer should be preallocated large enough to store the log records.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.10 UF_SetCustomLogField

There is a 4 byte reserved field in each log record. This function is used for setting this custom value. If the custom field is not set, it will be filled with NULL.

Function Prototype

```
UF_RET_CODE UF_SetCustomLogField( UF_LOG_SOURCE source, unsigned customField );
```

Parameters

source

Users can set 4 different custom values according to the source of log records.

Source	Description
UF_LOG_SOURCE_OTHER	If the log is generated by Packet Protocol commands or freescan, this value will be used.
UF_LOG_SOURCE_IN0	If the log is generated by an Input port, the respective values will be used.
UF_LOG_SOURCE_IN1	
UF_LOG_SOURCE_IN2	

customField

4 byte custom value.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.17.11 UF_GetCustomLogField

Reads the custom value of the specified log source.

Function Prototype

```
UF_RET_CODE UF_GetCustomLogField( UF_LOG_SOURCE source, unsigned* customField );
```

Parameters

source

Log source.

customField

Pointer to the 4 byte custom value to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18 Extended Wiegand API

Extended Wiegand Interface supports up to 64 bit Wiegand formats. The only constraint is that the ID field is limited to 32 bits. It also supports advanced options such as Fail ID and Inverse Parity on Fail.

- UF_SetWiegandFormat: configures the Wiegand format.
- UF_GetWiegandFormat: gets the Wiegand format information.
- UF_SetWiegandIO: configures the Wiegand IO ports.
- UF_GetWiegandIO: gets the configurations of the Wiegand IO ports.
- UF_SetWiegandOption: sets the advanced options.
- UF_GetWiegandOption: gets the advanced options.
- UF_SetAltValue: sets the alternative value of a field.
- UF_ClearAltValue: clears the alternative value of a field.
- UF_GetAltValue: gets the alternative value of a field.
- UF_MakeWiegandConfiguration: makes Wiegand configuration data to be saved into a file.

3.18.1 UF_SetWiegandFormat

Configures the Wiegand format.

Function Prototype

```
UF_RET_CODE    UF_SetWiegandFormat(    UFWiegandFormatHeader*    header,  
UFWiegandFormatData* data, int pulseWidth, int pulseInterval );
```

Parameters

header

UFWiegandFormatHeader is defined as follows;

```
typedef struct {  
    UF_WIEGAND_FORMAT format; // UF_WIEGAND_26BIT,  
                               // UF_WIEGAND_PASS_THRU,  
                               // UF_WIEGAND_CUSTOM  
  
    int totalBits;  
} UFWiegandFormatHeader;
```

data

Wiegand format data. If the format is UF_WIEGAND_26BIT, there is no format data and this parameter will be ignored. UFWiegandFormatData is defined as follows;

```
typedef struct {  
    int numOfIDField;  
    UFWiegandField  field[MAX_WIEGAND_FIELD];  
} UFWiegandPassThruData;
```

```
typedef struct {  
    int numOfField;  
    UINT32 idFieldMask;  
    UFWiegandField  field[MAX_WIEGAND_FIELD];  
    int numOfParity;
```



```
    UFWiegandParity parity[MAX_WIEGAND_PARITY];  
} UFWiegandCustomData;
```

```
typedef union {  
    UFWiegandPassThruData passThruData;  
    UFWiegandCustomData customData;  
} UFWiegandFormatData;
```

pulseWidth

Specifies the width of Wiegand signal.

pulseInterval

Specifies the interval of Wiegand signal.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.2 UF_GetWiegandFormat

Gets the Wiegand format data.

Function Prototype

```
UF_RET_CODE    UF_GetWiegandFormat(    UFWiegandFormatHeader*    header,  
UFWiegandFormatData* data,    int* pulseWidth,    int* pulseInterval );
```

Parameters

header

Pointer to the format header to be returned.

data

Pointer to the format data to be returned.

pulseWidth

Pointer to the width of Wiegand signal.

pulseInterval

Pointer to the interval of Wiegand signal.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.3 UF_SetWiegandIO

Configures the Wiegand IO ports.

Function Prototype

```
UF_RET_CODE    UF_SetWiegandIO(    UF_WIEGAND_INPUT_MODE    inputMode,  
UF_WIEGAND_OUTPUT_MODE outputMode, int numOfChar );
```

Parameters

inputMode

Mode	Description
UF_WIEGAND_INPUT_DISABLE	Ignores Wiegand inputs.
UF_WIEGAND_INPUT_VERIFY	Starts verification after receiving Wiegand inputs.

outputMode

Mode	Description
UF_WIEGAND_OUTPUT_DISABLE	Disables Wiegand output.
UF_WIEGAND_OUTPUT_WIEGAND_ONLY	Outputs Wiegand signal only if the verification is initiated by Wiegand input.
UF_WIEGAND_OUTPUT_ALL	Outputs Wiegand signal if matching succeeds.
UF_WIEGAND_OUTPUT_ABA_TRACK_II	Outputs ABA Track II characters instead of Wiegand signal.

numOfChar

Number of characters in ABA Track II output format. It is ignored if outputMode is not UF_WIEGAND_OUTPUT_ABA_TRACK_II.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.4 UF_GetWiegandIO

Gets the configurations of Wiegand IO ports.

Function Prototype

```
UF_RET_CODE    UF_GetWiegandIO(    UF_WIEGAND_INPUT_MODE*    inputMode,  
UF_WIEGAND_OUTPUT_MODE* outputMode, int* numOfChar );
```

Parameters

inputMode

Pointer to Wiegand input mode to be returned.

outputMode

Pointer to Wiegand output mode to be returned.

numOfChar

Pointer to the number of characters in ABA Track II output format. It is ignored if outputMode is not UF_WIEGAND_OUTPUT_ABA_TRACK_II.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.5 UF_SetWiegandOption

Sets advanced options.

Function Prototype

```
UF_RET_CODE UF_SetWiegandOption( BOOL useFailID, UINT32 failID, BOOL  
inverseParityOnFail );
```

Parameters

useFailID

Normally the module outputs Wiegand signal only if matching succeeds. If this option is TRUE, the module outputs the fail ID when matching fails.

failID

ID to be output if useFailID is TRUE.

inverseParityOnFail

If this option is TRUE, the module outputs Wiegand signals with inverted parities when matching fails.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.6 UF_GetWiegandOption

Gets the advanced options.

Function Prototype

```
UF_RET_CODE UF_GetWiegandOption( BOOL* useFailID, UINT32* failID,  
    BOOL* inverseParityOnFail );
```

Parameters

userFailID

Pointer to the useFailID option to be returned.

failID

Pointer to the fail ID to be returned.

inverseParityOnFail

Pointer to the inverseParityOnFail option to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.7 UF_SetAltValue

If the Wiegand format is UF_WIEGAND_26BIT or UF_WIEGAND_CUSTOM, users can set alternative values for non-ID fields. If an alternative value is set for a non-ID field, the module will replace the field with the alternative value before outputting the signal.

Function Prototype

```
UF_RET_CODE UF_SetAltValue( int fieldIndex, UINT32 value );
```

Parameters

fieldIndex

Index of the field.

value

Alternative value of the field.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.8 UF_ClearAltValue

Clears the alternative value of a field.

Function Prototype

```
UF_RET_CODE UF_ClearAltValue( int fieldIndex );
```

Parameters

fieldIndex

Index of the field.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.18.9 UF_GetAltValue

Gets the alternative value of a field.

Function Prototype

```
UF_RET_CODE UF_GetAltValue( int fieldIndex, UINT32* value );
```

Parameters

fieldIndex

Index of the field.

value

Alternative value of the field to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. If alternative value is not set to the field, return UF_ERR_NOT_FOUND. Otherwise, return the corresponding error code.

3.18.10 UF_MakeWiegandConfiguration

Makes Wiegand configuration data to be saved into a file.

Function Prototype

```
UF_RET_CODE    UF_MakeWiegandConfiguration(    UFConfigComponentHeader*  
configHeader, BYTE* configData );
```

Parameters

configHeader

Pointer to the configuration header to be returned.

configData

Pointer to the configuration data to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

See CUniFingerUIWiegandView::OnWiegandSaveFile in UniFingerUI source codes.

3.19 Wiegand Command Card API

Users can map an input function to a Wiegand ID. When the module detects the mapped IDs in Wiegand input port, it will execute the corresponding input function.

- `UF_AddWiegandCommandCard`: adds a mapping of input function to the specified ID.
- `UF_GetWiegandCommandCardList`: gets all the mapping data.
- `UF_ClearAllWiegandCommandCard`: clears all the mappings.

3.19.1 UF_AddWiegandCommandCard

Maps the input function to the specified ID.

Function Prototype

```
UF_RET_CODE UF_AddWiegandCommandCard( UINT32 userID, UF_INPUT_FUNC
function );
```

Parameters

userID

User ID.

function

Among the input functions, the followings can be assigned to command cards.

Function	Description
UF_INPUT_ENROLL	Enroll
UF_INPUT_IDENTIFY	Identify
UF_INPUT_DELETE	Delete
UF_INPUT_DELETE_ALL	Delete all
UF_INPUT_ENROLL_BY_WIEGAND	Enroll by Wiegand ID
UF_INPUT_DELETE_BY_WIEGAND	Delete by Wiegand ID
UF_INPUT_ENROLL_VERIFICATION	Enroll after administrator's verification
UF_INPUT_ENROLL_BY_WIEGAND _VERIFICATION	Enroll by Wiegand ID after administrator's verification
UF_INPUT_DELETE_VERIFICATION	Delete after administrator's verification
UF_INPUT_DELETE_BY_WIEGAND _VERIFICATION	Delete by Wiegand ID after administrator's verification
UF_INPUT_DELETE_ALL _VERIFICATION	Delete all after administrator's verification

Return Values

If the function succeeds, return `UF_RET_SUCCESS`. Otherwise, return the corresponding error code.

3.19.2 UF_GetWiegandCommandCardList

Gets the list of all the command cards.

Function Prototype

```
UF_RET_CODE      UF_GetWiegandCommandCardList(      int*      numOfCard,
UFWiegandCommandCard* commandCard );
```

Parameters

numOfCard

Number of command cards to be returned.

commandCard

Array of command card information to be returned. UFWiegandCommandCard is defined as follows;

```
typedef struct {
    UINT32      userID;
    UF_INPUT_FUNC      function;
} UFWiegandCommandCard;
```

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.19.3 UF_ClearAllWiegandCommandCard

Clears all the command card mappings.

Function Prototype

```
UF_RET_CODE UF_ClearAllWiegandCommandCard();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20 SmartCard API

BioEntry Smart readers support MIFARE types of smartcards. These functions provide basic functionalities such as read, write, and format smartcards.

- `UF_ReadSmartCard`: reads a smartcard.
- `UF_ReadSmartCardWithAG`: reads a smartcard with access group information.
- `UF_WriteSmartCard`: writes templates into a smartcard.
- `UF_WriteSmartCardWithAG`: writes templates and access group information into a smartcard.
- `UF_WriteSmartCardWithEntranceLimit`: writes templates, access group information, and entrance limit into a smartcard.
- `UF_FormatSmartCard`: formats a smartcard.
- `UF_SetSmartCardMode`: sets the operation mode.
- `UF_GetSmartCardMode`: gets the operation mode.
- `UF_ChangePrimaryKey`: changes the primary key.
- `UF_ChangeSecondaryKey`: changes the secondary key.
- `UF_SetKeyOption`: sets the site key options.
- `UF_GetKeyOption`: gets the site key options.
- `UF_SetCardLayout`: sets the layout of smartcard.
- `UF_GetCardLayout`: gets the layout of smartcard.
- `UF_SetSmartCardCallback`: sets the callback function for smartcard operation.

3.20.1 UF_ReadSmartCard

Reads a smart card.

Function Prototype

```
UF_RET_CODE UF_ReadSmartCard( UFCardHeader* header, BYTE* template1,  
BYTE* template2 );
```

Parameters

header

UFCardHeader is defined as follows;

```
typedef struct {  
    UINT32 csn; // 4 byte card serial number  
    UINT32 wiegandLower; // lower 4 bytes of Wiegand string  
    UINT32 wiegandHigher; // higher 4 bytes of Wiegand string  
    BYTE version;  
    BYTE commandType; // reserved for command cards. In V1.7 or  
                      // later firmware, this field is used for  
                      // storing entrance limit for non-command cards.  
  
    BYTE securityLevel; // security level of the user  
    BYTE numOfTemplate; // number of templates stored in the card  
    BYTE template1Duress; // 1 if the template1 is  
                          // of duress finger  
    BYTE template1Length[2]; // length of template1  
    BYTE template2Duress;  
    BYTE template2Length[2];  
} UFCardHeader;
```

template1

Pointer to the first template data read from the smartcard.

template2

Pointer to the second template data read from the smartcard.

Return Values

If the function succeeds, return `UF_RET_SUCCESS`. Otherwise, return the corresponding error code.

3.20.2 UF_ReadSmartCardWithAG

Reads a smart card with access group information. As for access group, see UF_AddAccessGroup.

Function Prototype

```
UF_RET_CODE  UF_ReadSmartCardWithAG(  UFCardHeader*  header,  BYTE*  
template1, BYTE* template2, int* numOfAccessGroup, BYTE* accessGroup );
```

Parameters

header

Pointer to the UFCardHeader to be returned.

template1

Pointer to the first template data read from the smartcard.

template2

Pointer to the second template data read from the smartcard.

numOfAccessGroup

Pointer to the number of access groups assigned to the smartcard.

accessGroup

Pointer to the IDs of access groups assigned to the smartcard. The length of each ID is 1 byte.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.3 UF_WriteSmartCard

Writes template data and header information into a smartcard.

Function Prototype

```
UF_RET_CODE UF_WriteSmartCard( UINT32 userID, UF_CARD_SECURITY_LEVEL
securityLevel, int numOfTemplate, int templateSize, BYTE* template1,
BOOL duress1, BYTE* template2, BOOL duress2 );
```

Parameters

userID

User ID.

securityLevel

Security level. If it is set to UF_SECURITY_READER_DEFAULT, the security level is same as defined in the BioEntry reader. If it is set to UF_SECURITY_BYPASS, the BioEntry reader will bypass the fingerprint authentication.

numOfTemplate

Number of templates to be written.

templateSize

Size of a template.

template1

Pointer to the first template data.

duress1

Specifies if the first template is of a duress finger.

template2

Pointer to the second template data.

duress2

Specifies if the second template is of a duress finger.

Return Values

If the function succeeds, return `UF_RET_SUCCESS`. Otherwise, return the corresponding error code.

3.20.4 UF_WriteSmartCardWithAG

Writes template data and access group information into a smartcard.

Function Prototype

```
UF_RET_CODE      UF_WriteSmartCardWithAG(      UINT32      userID,  
UF_CARD_SECURITY_LEVEL  securityLevel,  int  numOfTemplate,      int  
templateSize, BYTE* template1, BOOL duress1,  BYTE* template2, BOOL  
duress2, int numOfAccessGroup, BYTE* accessGroup );
```

Parameters

userID

User ID.

securityLevel

Security level. If it is set to UF_SECURITY_READER_DEFAULT, the security level is same as defined in the BioEntry reader. If it is set to UF_SECURITY_BYPASS, the BioEntry reader will bypass the fingerprint authentication.

numOfTemplate

Number of templates to be written.

templateSize

Size of a template.

template1

Pointer to the first template data.

duress1

Specifies if the first template is of a duress finger.

template2

Pointer to the second template data.

duress2

Specifies if the second template is of a duress finger.

numOfAccessGroup

Number of access groups assigned to the smartcard.

accessGroup

Pointer to the IDs of access group to be written.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.5 UF_WriteSmartCardWithEntranceLimit

Writes template data, access group, and entrance limit information into a smartcard.

Function Prototype

```
UF_RET_CODE    UF_WriteSmartCardWithEntranceLimit(    UINT32    userID,  
UF_CARD_SECURITY_LEVEL    securityLevel,    int    entranceLimit,    int  
numOfTemplate,    int    templateSize, BYTE*    template1, BOOL    duress1,    BYTE*  
template2,    BOOL    duress2,    int    numOfAccessGroup, BYTE*    accessGroup );
```

Parameters

userID

User ID.

securityLevel

Security level. If it is set to UF_SECURITY_READER_DEFAULT, the security level is same as defined in the BioEntry reader. If it is set to UF_SECURITY_BYPASS, the BioEntry reader will bypass the fingerprint authentication.

entranceLimit

Entrance limit between 0 and 7.

numOfTemplate

Number of templates to be written.

templateSize

Size of a template.

template1

Pointer to the first template data.

duress1

Specifies if the first template is of a duress finger.

template2

Pointer to the second template data.

duress2

Specifies if the second template is of a duress finger.

numOfAccessGroup

Number of access groups assigned to the smartcard.

accessGroup

Pointer to the IDs of access group to be written.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.6 UF_FormatSmartCard

Formats a smartcard.

Function Prototype

```
UF_RET_CODE UF_FormatSmartCard( BOOL templateOnly );
```

Parameters

templateOnly

If TRUE, erases only the template area and don't change the header information. If FALSE, erase the header information, too.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.7 UF_SetSmartCardMode

Sets the operation mode of the BioEntry Smart reader.

Function Prototype

```
UF_RET_CODE UF_SetSmartCardMode( UF_CARD_MODE mode );
```

Parameters

mode

Operation mode.

Mode	Description
UF_CARD_DISABLE	Disables the smartcard operation.
UF_CARD_VERIFY_ID	After detecting a smartcard, verifies the fingerprint input with the templates stored in the BioEntry reader.
UF_CARD_VERIFY_TEMPLATE	After detecting a smartcard, verifies the fingerprint input with the templates stored in the card.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.8 UF_GetSmartCardMode

Gets the operation mode.

Function Prototype

```
UF_RET_CODE UF_GetSmartCardMode( UF_CARD_MODE* mode );
```

Parameters

mode

Pointer to the operation mode to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.9 UF_ChangePrimaryKey

To prevent illegal access, MIFARE card is encrypted using 48bit site key. The site key should be handled with utmost caution. If it is disclosed, the data on the smartcard will not be secure any more. UF_ChangePrimaryKey is used to change the primary site key.

Function Prototype

```
UF_RET_CODE    UF_ChangePrimaryKey(    BYTE*    oldPrimaryKey,    BYTE*  
newPrimaryKey );
```

Parameters

oldPrimaryKey

Pointer to the old site key.

newPrimaryKey

Pointer to the new site key.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.10 UF_ChangeSecondaryKey

Changes the secondary site key. The secondary site key is used only when the useSecondaryKey option is set by UF_SetKeyOption.

Function Prototype

```
UF_RET_CODE    UF_ChangeSecondaryKey(    BYTE*    primaryKey,    BYTE*  
newSecondaryKey );
```

Parameters

primaryKey

Pointer to the primary key.

newSecondaryKey

Pointer to the new secondary key.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.11 UF_SetKeyOption

When changing the site key, BioEntry readers have to handle cards with new site key and cards with old site key at the same time. In that case, useSecondaryKey option can be used. If the secondary key is set to old site key, the reader will handle both types of cards. If autoUpdate option is on, the reader automatically replaces the old site key with new one whenever detecting a smartcard with old key.

Function Prototype

```
UF_RET_CODE UF_SetKeyOption( BYTE* primaryKey, BOOL useSecondaryKey,  
BOOL autoUpdate );
```

Parameters

primaryKey

Pointer to the primary key.

useSecondaryKey

If TRUE, process the cards encrypted with the secondary key.

autoUpdate

If TRUE, replace the secondary key with primary key when detecting a smartcard encrypted with the secondary key.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.12 UF_GetKeyOption

Gets the site key options.

Function Prototype

```
UF_RET_CODE    UF_GetKeyOption(    BOOL*    useSecondaryKey,    BOOL*  
autoUpdate );
```

Parameters

useSecondaryKey

Pointer to userSecondaryKey option.

autoUpdate

Pointer to autoUpdate option.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.13 UF_SetCardLayout

Changes the layout of the smartcard. By default, a smartcard stores two fingerprint templates. If there are not sufficient spaces on the card, or some blocks of it are reserved for other data, users can change the layout using this function. Changing card layout should be handled with utmost caution. If you aren't sure what to do, contact to support@supremainc.com first before trying yourself.

Function Prototype

```
UF_RET_CODE UF_SetCardLayout( UFCarLayout* layout );
```

Parameters

layout

Pointer to the layout information. UFCarLayout is defined as follows;

```
typedef struct {  
    unsigned short templateSize;  
  
    BYTE headerBlock;  
    BYTE template1StartBlock;  
    BYTE template1BlockSize;  
    BYTE template2StartBlock;  
    BYTE template2BlockSize;  
} UFCarLayout;
```

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.14 UF_GetCardLayout

Gets the card layout information.

Function Prototype

```
UF_RET_CODE UF_GetCardLayout( UFCardLayout* layout );
```

Parameters

layout

Pointer to UFCardLayout.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.20.15 UF_SetSmartCardCallback

Sets the callback function for smartcard operation. This callback is called after scanning a smartcard successfully.

Function Prototype

```
void UF_SetSmartCardCallback( void (*callback) ( BYTE ) );
```

Parameters

callback

Pointer to the callback function.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21 Access Control API

Since V1.6 firmware, BioEntry readers provide access control features such as time schedule and access group. By using these functions, user's access can be controlled in finer detail.

- `UF_AddTimeSchedule`: adds a time schedule.
- `UF_GetTimeSchedule`: reads the specified time schedule.
- `UF_DeleteTimeSchedule`: deletes a time schedule.
- `UF_DeleteAllTimeSchedule`: deletes all time schedules.
- `UF_AddHoliday`: adds a holiday schedule.
- `UF_GetHoliday`: reads the specified holiday schedule.
- `UF_DeleteHoliday`: deletes a holiday schedule.
- `UF_DeleteAllHoliday`: deletes all holiday schedules.
- `UF_AddAccessGroup`: adds an access group.
- `UF_GetAccessGroup`: reads the specified access group.
- `UF_DeleteAccessGroup`: deletes an access group.
- `UF_DeleteAllAccessGroup`: deletes all access groups.
- `UF_SetUserAccessGroup`: assigns access groups to a user.
- `UF_GetUserAccessGroup`: gets the access groups of a user.

3.21.1 UF_AddTimeSchedule

A BioEntry reader can store up to 64 time schedules. Each time schedule consists of 7 daily schedules and an optional holiday schedule. And each daily schedule may have up to 5 time segments.

```
#define UF_TIMECODE_PER_DAY      5

typedef struct {
    unsigned short startTime; // start time in minutes
    unsigned short endTime;  // end time in minutes
} UTimeCodeElem;

typedef struct {
    UTimeCodeElem codeElement[UF_TIMECODE_PER_DAY];
} UTimeCode;

typedef struct {
    int scheduleID;
    UTimeCode timeCode[7]; // 0 - Sunday, 1 - Monday, ...
    int holidayID;
} UTimeSchedule;
```

Function Prototype

```
UF_RET_CODE UF_AddTimeSchedule( UTimeSchedule* schedule );
```

Parameters

schedule

Pointer to the time schedule to be added.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the

corresponding error code.

Example

```
UFTimeSchedule timeSchedule;

memset( &timeSchedule, 0, sizeof(UFTimeSchedule) ); // clear the
structure

timeSchedule.scheduleID = 1;
timeSchedule.holidayID = 1;

// Monday- 09:00 ~ 18:00
timeSchedule.timeCode[1].codeElement[0].startTime = 9 * 60;
timeSchedule.timeCode[1].codeElement[0].endTime = 18 * 60;

// Tuesday- 08:00 ~ 12:00 and 14:30 ~ 20:00
timeSchedule.timeCode[2].codeElement[0].startTime = 8 * 60;
timeSchedule.timeCode[2].codeElement[0].endTime = 12 * 60;
timeSchedule.timeCode[2].codeElement[1].startTime = 14 * 60 + 30;
timeSchedule.timeCode[2].codeElement[1].endTime = 20 * 60;

// ...

UF_RET_CODE result = UF_AddTimeSchedule( &timeSchedule );
```

3.21.2 UF_GetTimeSchedule

Reads the specified time schedule.

Function Prototype

```
UF_RET_CODE UF_GetTimeSchedule( int ID, UFTimeSchedule* schedule );
```

Parameters

ID

ID of the time schedule.

schedule

Pointer to the time schedule to be read.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.3 UF_DeleteTimeSchedule

Deletes the specified time schedule.

Function Prototype

```
UF_RET_CODE UF_DeleteTimeSchedule( int ID );
```

Parameters

ID

ID of the time schedule.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.4 UF_DeleteAllTimeSchedule

Deletes all the time schedules stored in a BioEntry reader.

Function Prototype

```
UF_RET_CODE UF_DeleteAllTimeSchedule();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.5 UF_AddHoliday

Each time schedule may have an optional holiday schedule. A holiday schedule consists of a holiday list and a daily schedule for it.

```
typedef struct {
    int holidayID;
    int numOfHoliday;
    unsigned short holiday[32]; // (month < 8) | day
    UFTIME_CODE timeCode;
} UFHoliday;
```

Function Prototype

```
UF_RET_CODE UF_AddHoliday( UFHoliday* holiday );
```

Parameters

holiday

Pointer to the holiday schedule to be added.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example

```
UFHoliday holiday;

memset( &holiday, 0, sizeof(UFHoliday) ); // clear the structure

holiday.holidayID = 1;
holiday.numOfHoliday = 10;
```

```
// Jan. 1 is holiday
holiday.holiday[0] = (1 << 8) | 1;

// Mar. 5 is holiday
holiday.holiday[1] = (3 << 8) | 5;

// ...

// Access is granted during 09:00 ~ 10:00 on holideys
holiday.timeCode.codeElement[0].startTime = 9 * 60;
holiday.timeCode.codeElement[0].endTime = 10 * 60;

UF_RET_CODE result = UF_AddHoliday( &holiday );
```

3.21.6 UF_GetHoliday

Reads the specified holiday schedule.

Function Prototype

```
UF_RET_CODE UF_GetHoliday( int ID, UFHoliday* holiday );
```

Parameters

ID

ID of the holiday schedule.

holiday

Pointer to the holiday schedule to be read.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.7 UF_DeleteHoliday

Deletes the specified holiday schedule.

Function Prototype

```
UF_RET_CODE UF_DeleteHoliday( int ID );
```

Parameters

ID

ID of the holiday schedule.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.8 UF_DeleteAllHoliday

Deletes all the holiday schedules stored in a BioEntry reader.

Function Prototype

```
UF_RET_CODE UF_DeleteAllHoliday();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.9 UF_AddAccessGroup

Each access group may have up to 16 time schedules. The access of members is granted only when the time belongs to the time schedules of the group.

```
#define UF_SCHEDULE_PER_GROUP    16

typedef struct {
    int groupID;
    int numOfSchedule;
    int scheduleID[UF_SCHEDULE_PER_GROUP];
} UFAccessGroup;
```

Function Prototype

```
UF_RET_CODE UF_AddAccessGroup( UFAccessGroup* group );
```

Parameters

group

Pointer to the access group to be added.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.10 UF_GetAccessGroup

Reads the specified access group.

Function Prototype

```
UF_RET_CODE UF_GetAccessGroup( int ID, UFAccessGroup* group );
```

Parameters

ID

ID of the access group.

group

Pointer to the access group to be read.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.11 UF_DeleteAccessGroup

Deletes the specified access group.

Function Prototype

```
UF_RET_CODE UF_DeleteAccessGroup( int ID );
```

Parameters

ID

ID of the access group.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.12 UF_DeleteAllAccessGroup

Deletes all the access groups stored in a BioEntry reader.

Function Prototype

```
UF_RET_CODE UF_DeleteAllAccessGroup();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.13 UF_SetUserAccessGroup

Assigns access groups to a user. A user can be a member of up to 4 access groups.

Function Prototype

```
UF_RET_CODE UF_SetUserAccessGroup( UINT32 userID, int numOfGroup,  
int* groupID );
```

Parameters

userID

User ID.

numOfGroup

Number of access groups to be assigned.

groupID

Array of access group IDs to be assigned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.21.14 UF_GetUserAccessGroup

Reads the IDs of access groups assigned to a user.

Function Prototype

```
UF_RET_CODE UF_GetUserAccessGroup( UINT32 userID, int* numOfGroup,  
int* groupID );
```

Parameters

userID

User ID.

numOfGroup

Pointer to the number of access groups to be returned.

groupID

Array of access group IDs to be returned.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.22 Blacklist API

When a user ID is added to the blacklist, authentication will always fail regardless of fingerprint matching result. The blacklist takes precedence over the authentication type of a user. For example, though the authentication mode of a user is UF_AUTH_BYPASS, the authentication would fail if it is in the blacklist. The blacklist can store up to 1022 user IDs.

- UF_AddBlacklist: adds an ID to the blacklist.
- UF_DeleteBlacklist: deletes an ID from the blacklist.
- UF_GetBlacklist: reads the IDs in the blacklist.
- UF_DeleteAllBlacklist: clears the blacklist.

3.22.1 UF_AddBlacklist

Adds a user ID to the blacklist.

Function Prototype

```
UF_RET_CODE UF_AddBlacklist( UINT32 userID, int* numOfBlacklistedID );
```

Parameters

userID

User ID.

numOfBlacklistedID

Number of IDs in the blacklist after adding.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.22.2 UF_DeleteBlacklist

Deletes an ID from the blacklist.

Function Prototype

```
UF_RET_CODE      UF_DeleteBlacklist(      UINT32      userID,      int*  
numOfBlacklistedID );
```

Parameters

userID

User ID.

numOfBlacklistedID

Number of IDs in the blacklist after deleting.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.22.3 UF_GetBlacklist

Receive user IDs in the blacklist.

Function Prototype

```
UF_RET_CODE  UF_GetBlacklist(  int*    numOfBlacklistedID,  UINT32*  
userID );
```

Parameters

numOfBlacklistedID

Pointer to the number of IDs in the blacklist.

userID

Array of user IDs in the blacklist. This should be pre-allocated large enough.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.22.4 UF_DeleteAllBlacklist

Clears the blacklist.

Function Prototype

```
UF_RET_CODE UF_DeleteAllBlacklist();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.23 WSQ API

The Wavelet Scalar Quantization (WSQ) Gray-scale Fingerprint Image Compression Algorithm is the standard for the exchange of 8-bit, 500ppi fingerprint images within the criminal justice community. SFM5500 / SFM6000 / SFMSlim series provide the WSQ compression feature when the module scanning an image.

(SFM5500 / 6000 / Slim series are supported.)

Suprema's WSQ algorithm is certified by FBI.

(<https://www.fbibiospecs.cjis.gov/WSQ/Implementations>)

- UF_ScanImageEx: Scans a fingerprint on the sensor and retrieves the image data.
- UF_ReadImageEx : Retrieves the last scanned fingerprint image.
- UF_WSQ_Decode : Decode the compressed WSQ fingerprint image which is read or scanned by the module.

3.23.1 UF_ScanImageEX

Scans a fingerprint on the sensor and retrieves the image data.

Function Prototype

```
UF_RET_CODE UF_ScanImageEx( UFIImage* image, UF_IMAGE_TYPE type, int
wsqBitRate );
```

Parameters

image

Pointer to the UFIImage structure.

type

Type of an image to read or scan.

wsqBitRate

WSQ is compressed by input of wsqBitRate in the module. The value of wsqBitRate is recommended as below.

1. *High quality compressing (compressing rate approx. 1:5)*
wsqBitRate = 225,
2. *Midium quality compressing (compressing rate approx. 1:10)*
wsqBitRate = 150,
3. *Low quality compressing (compressing rate approx. 1:15)*
wsqBitRate = 75,

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.23.2 UF_ReadImageEx

Retrieves the last scanned fingerprint image.

Function Prototype

```
UF_RET_CODE UF_ReadImageEx( UFIImage* image, UF_IMAGE_TYPE type, int  
wsqBitRate );
```

Parameters

image

Pointer to the UFIImage structure.

type

Type of an image to read or scan.

wsqBitRate

WSQ is compressed by input of wsqBitRate in the module. The value of wsqBitRate is recommended as below.

4. *High quality compressing (compressing rate approx. 1:5)*
wsqBitRate = 225,
5. *Midium quality compressing (compressing rate approx. 1:10)*
wsqBitRate = 150,
6. *Low quality compressing (compressing rate approx. 1:15)*
wsqBitRate = 75,

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.23.3 UF_WSQ_Decode

Decode the compressed WSQ fingerprint image which is read or scanned by the module.

Function Prototype

```
UF_RET_CODE UF_WSQ_Decode(unsigned char **odata, int *ow, int *oh,  
int *od, int *oppi, int *lossyflag, unsigned char *idata, const int  
ilen);
```

Parameters

odata

Pointer to the decoded image data.

ow

Pointer to the width of decoded image data.

oh

Pointer to the height of decoded image data.

od

Pointer to the bit depth of decoded image data. (Always returns 8)

lossyflag

Pointer to the lossy flag. (Always returns 1)

idata

Pointer to the read or scanned WSQ fingerprint image data by the module.

ilen

Pointer to the length of idata

odata is dynamic allocated by the UF_WSQ_Decode function. You should free memory of odata after use it.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

Example code

```
void CUniFingerUIView::DrawFingerprintImage()
{
    BOOL bCompressed = m_Image->compressed;
    int nSize = m_Image->imgLen;

    if(bCompressed == UF_WSQ_IMAGE)
    {
        unsigned char *sourceBuffer = (BYTE*)m_Image->buffer;
        unsigned char *WSQData = NULL;

        int ow;
        int oh;
        int od;
        int oppi;
        int lossyflag;

        if(UF_WSQ_Decode(&WSQData, &ow, &oh, &od, &oppi, &lossyflag,
sourceBuffer, nSize)==UF_RET_SUCCESS)
        {
            memcpy(sourceBuffer,WSQData, ow*oh);
            if(WSQData)
                free(WSQData);
        }
    }
    m_Fingerprint.SetImage( m_Image );
    m_Fingerprint.Invalidate();
}
```

3.24 File System API

- UF_ResetSystemConfiguration : Reset all system parameter and GPIO configuration of the module as factory default.
- UF_FormatUserDatabase : Erase the user database partition (which is specific area stores fingerprint template into flash memory.) of the module. All templates are erased and filled with 0xFF .

3.24.1 UF_ResetSystemConfiguration

Reset all system parameter and GPIO configuration of the module as factory default.

Function Prototype

```
UF_RET_CODE UF_ResetSystemConfiguration();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.24.2 UF_FormatUserDatabase

Erase the user database partition (which is specific area stores fingerprint template into flash memory.) of the module.

Function Prototype

```
UF_RET_CODE UF_FormatUserDatabase();
```

Parameters

None

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.25 Secure Packet Protocol API

SFM6000 / SFMSlim series provide the secure packet protocol to prevent external unpermitted attacks such as sniffing. This feature will be helpful in applications that require strong security. The details about the secure packet protocol are described in a document called SFM Packet Protocol Manual. Please refer to the document for more details.

(SFM6000/Slim series are supported.)

- `UF_GetSecurePacketProtocolMode` : Gets whether secure packet protocol mode is enabled or not.
- `UF_SetSecurePacketProtocolMode` : Sets the secure packet protocol mode enable or disable.
- `UF_SetSecureCode` : Sets the secure code into the secure packet.
- `UF_CreateRandomSecureKey` : Creates a random secure key for the secure packet protocol. This key is used to communicate using a temporary encrypted secure packet.
- `UF_CreateKeyPair` : Creates a key pair (private key and public key) for the public key exchange.
- `UF_GetSecureKey` : Gets a secure key from the public key of the module with the private key of the host.
- `UF_PublicKeyExchange` : Exchanges the public key from the host and the module to each other.

3.25.1 UF_GetSecurePacketProtocolMode

Gets whether secure packet protocol mode is enabled or not.

(SFM6000 / Slim series only supported.)

Function Prototype

```
BOOL UF_GetSecurePacketProtocolMode();
```

Parameters

None

Return Values

If the secure packet protocol mode is enabled return TRUE, Otherwise, return FALSE.

3.25.2 UF_SetSecurePacketProtocolMode

Sets the secure packet protocol mode enable or disable.

(SFM6000 / Slim series only supported.)

Function Prototype

```
BOOL  UF_GetSecurePacketProtocolMode (BOOL  securePacketProtocolMode,  
BYTE *secureKey);
```

Parameters

securePacketProtocolMode

If it is set to TRUE, the secure packet protocol is enabled. If the secure packet protocol is enabled, users can communicate via the encrypted packets.

secureKey

The secure key is an encryption key for packet encryption only. SFM6000 / Slim series have two kinds of encryption keys. The first one is for template encryption, and the second one is for packet encryption. For discriminating confusion of the terms, the key for packet encryption is named as secure key.

This key can be set 32byte length or NULL when the *securePacketProtocolMode* parameter is set to TRUE. If it is NULL, the secure key will be created randomly by using a public key exchange with SFM.

Return Values

If the secure packet protocol mode is changed successfully return TRUE, Otherwise, return FALSE.

3.25.3 UF_SetSecureCode

Sets the secure code into the secure packet.

(SFM6000 / Slim series only supported.)

Function Prototype

```
void UF_SetSecureCode (BYTE *secureCode);
```

Parameters

secureCode

The secure code is used to reach out to high-level security. If users use secure packet protocol, they can encrypt command protocol packets including this secure code. When users sent the encrypted command protocol packets with their own secure code, SFM returns the encrypted command protocol packets including the same secure code what users sent before. The secure code can be set 8 bytes length or NULL. If it is NULL, secure code will be randomly changed every packet communication pair about the request and response.

If the secure code from the SFM is different from the code that the users sent, it can be doubted as an attack from the strangers.

Return Values

None

3.25.4 UF_CreateKeyPair

Creates a key pair (private key and public key) for the public key exchange. These key pair is used to make a secure key for communicating between host and SFM. That is, when users use the secure packet protocol, they need a secure key for the command protocol packet encryption. The secure key can be randomly generated by using the public key exchange. Since FW v3.6 (SFM6000 / Slim series only), public key exchange based on ED25519 is supported.

(SFM6000 / Slim series only supported.)

Function Prototype

```
UF_RET_CODE      UF_CreateKeyPair (BYTE      *publicKey_host,      BYTE
*privateKey_host);
```

Parameters

publicKey_host

Pointer to 32byte length public key generated for the host

privateKey_host

Pointer to 64byte length private key generated for the host.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

3.25.5 UF_GetSecureKey

Gets a secure key from the public key of the module with the private key of the host.

This function is called in UF_CreateRandomSecureKey.

(SFM6000 / Slim series only supported.)

Function Prototype

```
void UF_GetSecureKey(BYTE *secureKey, BYTE *publicKey_module, BYTE  
*privateKey_host);
```

Parameters

secureKey

Pointer to 32byte length secure key generated from the publicKey_module and the privateKey_host that created by the UF_PublicKeyExchange and UF_CreateKeyPair, respectively. This key is used to encrypt the command protocol packets.

publicKey_module

Pointer to 32byte length public key generated from the module.

privateKey_host

Pointer to 64byte length private key generated from the host.

Return Values

None

3.25.6 UF_PublicKeyExchange

Exchanges the public key from the host and the module to each other. Users can get the public key generated from the module by using sending their public key generated from the host to the SFM. Also, users can get the secure key from their private key paired with the public key they sent via this function. Refer to UF_GetSecureKey, UF_CreateKeyPair.

(SFM6000 / Slim series only supported.)

Function Prototype

```
UF_RET_CODE    UF_PublicKeyExchange (BYTE    *publicKey_host,    BYTE
*publicKey_module);
```

Parameters

publicKey_host

Pointer to 32byte length public key generated from the host.

publicKey_module

Pointer to 32byte length public key generated from the module.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code

3.26 Key Management API

All of the keys not only the encryption key but also the secure key can be managed using this Key Management API.

The encryption key is used to encrypt the fingerprint template stored in the module. Also, The secure key is used to encrypt the packet for the secure packet protocol.

For more detail, the SFM6000 / SFMSlim series provide the secure packet protocol to prevent external unpermitted attacks such as sniffing. For this feature, the module requires the secure key for making the secure packet that is the encrypted packet. The secure key is another key from the encryption key which can encrypt the fingerprint template stored in the module as described before. however, if you don't set the secure key, the module will use the encryption key as the initial key for the secure packet protocol.

(SFM6000/Slim series are supported.)

- UF_ChangeKey : Changes keys such as the encryption key (including initialization vector in AES-CBC mode) and the secure key.

3.26.1 UF_ChangeKey

Changes keys such as the encryption key (including initialization vector in AES-CBC mode) and the secure key.

(SFM6000 / Slim only supported.)

Users can select an option below to change or reset the key.

Option	Description
UF_KEY_OPTION_SET_ENCRYPTION_KEY	Changes encryption key
UF_KEY_OPTION_SET_INITIALIZATION_VECTOR	Changes initialization vector in AES-CBC encryption mode
UF_KEY_OPTION_SET_SECURE_KEY	Changes secure key for the secure packet protocol
UF_KEY_OPTION_SET_ENCRYPTION_KEY_WITH_VERIFICATION	Changes encryption key with verification of the current encryption key already set.
UF_KEY_OPTION_SET_INITIALIZATION_VECTOR_WITH_VERIFICATION	Changes initialization vector with verification of the current initialization vector already set.
UF_KEY_OPTION_SET_SECURE_KEY_WITH_VERIFICATION	Changes secure key with verification of the current secure key already set
UF_KEY_OPTION_RESET_INITIALIZATION_VECTOR	Resets initialization vector as a default
UF_KEY_OPTION_RESET_SECURE_KEY	Resets secure key as a default
UF_KEY_OPTION_SET_PUBLIC_KEY	Sends the public key of the host
UF_KEY_OPTION_GET_PUBLIC_KEY	Receives the public key of the module

Function Prototype

```
UF_RET_CODE UF_ChangeKey(UF_KEY_OPTION option, BYTE *currentKey, BYTE *newKey, BOOL verification);
```

Parameters

option

An option to change the key.

currentKey

Currently set key in the module.

newKey

New key to change.

verification

If the value is TRUE, the *currentKey* and the *newKey* are required. And if the *currentKey* is not matched in the module, the key will not be changed.

If the value is FALSE, the *currentKey* is ignored. It means *newKey* will be set to the module forcibly.

Return Values

If the function succeeds, return UF_RET_SUCCESS. Otherwise, return the corresponding error code.

4 Contact Info

- **Headquarters**

Suprema, Inc. (<http://www.supremainc.com>)

17F Parkview Tower, 248, Jeongjail-ro, Bundang-gu,
Seongnam-si, Gyeonggi-do, 13554, Rep. of KOREA

Email: sales@supremainc.com