

U N I O N C O M M U N I T Y
S e r v e r S D K F o r W i n d o w s

P r o g r a m m e r ' s G u i d e
V e r s i o n 4 . 0

October 2009

Software Development Department

UNION COMMUNITY CO., LTD.

Copyright © 2008, UNION COMMUNITY Co., Ltd.

All rights reserved.

Union Community Co., Ltd.
Hyundai Topics Building, 44-3 Bangi-dong, Songpa-gu, Seoul 138-050 Korea
Tel: +82-6488-3000, Fax: +82-6488-3099 <http://www.unioncomm.co.kr>



USER License Agreement for Software Developer's Kit

Designed by UNION COMMUNITY Co., Ltd.

This agreement is a legal usage license agreement between Union Community Co., Ltd. and the user.

If you do not agree with the terms and conditions of the agreement, please return the product promptly.

If you return the product, you will receive a refund.

1. Usage License

UNION COMMUNITY Co., Ltd. grants the licensee a personal, limited, non-transferable license to use this SDK.

non-exclusive right to install and use one copy of the SDK on a single computer exclusively.

The software is considered 'being used' if it is stored in a computer's main or other storage device.

The number of software copies will be determined by taking the greater number of the number of computers 'used' by the software and the number of computers with the software installed.

The Licensee may use the SDK solely for developing, designing, and testing UNION software applications for Use with UNION products ("Applications").

2. Right to Upgrade

If you have purchased the software by upgrading an older version, the usage license of the old version

is transferred to the new version. However, you may only use the old version under the condition that

The old and new versions are not running simultaneously. Therefore, you are prohibited from transferring, renting or selling the old version. You retain the usage license for the program and ancillary files that are in the old version but not in the new version.

3. Assignment of License

If you wish to transfer the usage license of this software to a third party, you must first obtain written consent.

statement indicating that the recipient agrees to this agreement. You must then transfer the original

The disk and all other program components, along with all copies of the program, must be destroyed. After the

Once the transfer is complete, you must notify UNION COMMUNITY Co., Ltd. to update the customer registration.

The Licensee shall not rent, lease, sell, or lend the software application developed using the SDK to a third party. without the prior written consent of UNION.

The Licensee shall not copy and redistribute the SDK without UNION's prior written consent.

No other uses and/or distribution of the SDK or Sample Code are permitted without UNION's prior written consent.
written consent. UNION reserves all rights not expressly granted to Licensee.

4. Copyright

All copyrights and intellectual property rights of the software and its components belong to UNION.
COMMUNITY Co., Ltd. and these rights are protected under Korean and international copyright laws.
Therefore, you may not make copies of the software other than for your backup purposes. In addition,
You may not modify the software except for reverse-engineering purposes to ensure compatibility.
Finally, you may not modify, transform, or copy any part of the documentation without written permission.
permission from UNION COMMUNITY. (If you're using a network product, you may copy the
documentation in the amount of the number of users)

5. Installation

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

6. Limitation of Warranty

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

7. Liabilities

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

8. Termination

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

9. Customer Service

UNION COMMUNITY Co., Ltd. makes every effort to provide registered customers with technical assistance and solutions to problems regarding software applications under certain system environments.

When a customer submits a suggestion regarding any inconvenience or anomaly experienced during In the event of product usage issues, UNION COMMUNITY Co., Ltd. will take corrective action and notify the customer of the result.

10. General Terms

You acknowledge that you have read, understood, and agree to the terms of this agreement. You also Recognize that this agreement supersedes user agreements of older versions, past order agreements, advertisement notifications, and/or other written agreements.

11. Contact

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

Table of Contents

1. Overview	21
1.1 Application	21
1.2 Features	21
1.3 Development Environment	22
1.4 Module Configuration	22
1.5 Development Model	23
1.6 Glossary	24
2. Installation	31
2.1 System Requirements	31
2.2 Installation	32
2.3 Files to be Installed	36
2.3.1 Windows System Directory	36
2.3.2 GAC (Global Assembly Cache) Folder	36
2.3.3 (Installation Folder) \Bin	36
2.3.4 (Installation Folder) \dotNET	36
2.3.5 (Installation Folder)\dotNET\Setup	37
2.3.6 (Installation Folder) \ Inc	37
2.3.7 (Installation Folder) \ Lib	37
2.3.8 (Installation Folder) \Samples	37
2.3.9 (Installation Folder)\Skins	38
3. API Reference for DLL	39
3.1 Type definitions	39
3.1.1 Basic types	39
UCSAPI_SINT8 / UCSAPI_SINT16 / UCBioAPI_SINT32	39
UCSAPI_UINT8 / UCSAPI_UINT16 / UCBioAPI_UINT32	39
UCSAPI_SINT / UCSAPI_UINT	39
UCSAPI_VOID_PTR	39
UCSAPI_BOOL	39
UCSAPI_CHAR / UCSAPI_CHAR_PTR	39

UCSAPI_NULL	40
UCSAPI_HWND	40
3.1.2 General Types	40
UCSAPI_RETURN	40
UCSAPI_DATE_TIME_INFO	40
UCSAPI_MESSAGE	41
3.1.3 User information related types	41
UCSAPI_ACCESS_DATE_TYPE	41
UCSAPI_ACCESS_AUTHORITY	41
UCSAPI_CARD_DATA	42
UCSAPI_FINGER_DATA	42
UCSAPI_FACE_INFO	43
UCSAPI_FACE_DATA	44
UCSAPI_IRIS_INFO	44
UCSAPI_AUTH_DATA	45
UCSAPI_PICTURE_HEADER	45
UCSAPI_PICTURE_DATA	46
UCSAPI_USER_COUNT	46
UCSAPI_USER_INFO	47
UCSAPI_USER_DATA	48
UCSAPI_ACCESS_FLAG	49
UCSAPI_ERROR_TYPE	50
3.1.4 Log-related types	50
UCSAPI_GET_LOG_TYPE	50
UCSAPI_LOG_IMAGE	50
UCSAPI_ACCESS_LOG_DATA	51
3.1.5 Callback-related types	52
UCSAPI_CALLBACK_EVENT_HANDLER	52
UCSAPI_CALLBACK_PARAM_0	53
UCSAPI_PROGRESS_INFO	53
UCSAPI_CALLBACK_PARAM_1	54
UCSAPI_CALLBACK_DATA_TYPE	55
3.1.6 Access Control Setting Related Types	55
UCSAPI_TIMEZONE	55
UCSAPI_ACCESS_TIMEZONE	56

UCSAPI_ACCESS_TIMEZONE_DATA	56
UCSAPI_ACCESS_HOLIDAY	57
UCSAPI_ACCESS_HOLIDAY_DATA	57
UCSAPI_ACCESS_TIMEZONE_CODE	58
UCSAPI_ACCESS_TIME	58
UCSAPI_ACCESS_TIME_DATA	59
UCSAPI_ACCESS_GROUP	60
UCSAPI_ACCESS_GROUP_DATA	60
UCSAPI_ACCESS_CONTROL_DATA_TYPE	61
UCSAPI_ACCESS_CONTROL_DATA	61
3.1.7 Authentication-related types	62
UCSAPI_AUTH_TYPE	62
UCSAPI_AUTH_MODE	62
UCSAPI_INPUT_DATA_CARD	63
UCSAPI_INPUT_DATA_PASSWORD	63
UCSAPI_INPUT_DATA_FINGER_1_TO_1	64
UCSAPI_INPUT_DATA_FINGER_1_TO_N	65
UCSAPI_INPUT_DATA_TYPE	66
UCSAPI_INPUT_DATA	66
UCSAPI_INPUT_ID_TYPE	67
UCSAPI_INPUT_ID_DATA	68
UCSAPI_AUTH_INFO	68
UCSAPI_AUTH_NOFTY	69
3.1.8 Terminal Option Setting Related Types	70
UCSAPI_TERMINAL_TIMEZONE	70
UCSAPI_TERMINAL_DAY_SCHEDULE	71
UCSAPI_HOLIDAY_TYPE	72
UCSAPI_TERMINAL_HOLIDAY_INFO	72
UCSAPI_TERMINAL_SCHEDULE	73
UCSAPI_SECURITY_LEVEL	74
UCSAPI_ANTIPASSBACK_LEVEL	74
UCSAPI_NETWORK_INFO	75
UCSAPI_SERVER_INFO	76
UCSAPI_TERMINAL_OPTION_FLAG	76
UCSAPI_TERMINAL_OPTION	77

UCSAPI_ACU_OPTION	79
UCSAPI_ACU_LOCKSCHEDULE	80
UCSAPI_PERIPHERAL_DEVICE	81
UCSAPI_SET_EMERGENCY_INFO	81
UCSAPI_VOIP_INFO	82
3.1.9 Monitoring related types	83
UCSAPI_TERMINAL_STATUS	83
UCSAPI_ACU_STATUS_INFO	84
UCSAPI_TERMINAL_CONTROL	85
3.2 API References	87
3.2.1 General API	87
UCSAPI_ServerStart	87
UCSAPI_ServerStop	89
UCSAPI_SetTerminalTimezone	90
UCSAPI_SetError	92
UCSAPI_SetWiegandFormatToTerminal	93
UCSAPI_SetFingerImageSend	95
3.2.2 Terminal User Management API	96
UCSAPI_AddUserToTerminal	96
UCSAPI_DeleteUserFromTerminal	98
UCSAPI_DeleteAllUsersFromTerminal	100
UCSAPI_GetUserCountFromTerminal	101
UCSAPI_GetUserInfoListFromTerminal	102
UCSAPI_GetUserDataFromTerminal	103
UCSAPI_RegisterFaceFromTerminal	104
UCSAPI_RegisterWalkThroughFaceFromTerminal	105
UCSAPI_RegisterIrisFromTerminal	106
UCSAPI_EnrollFromTerminal	108
UCSAPI_FreeExportData	109
3.2.3 Log-related API	111
UCSAPI_GetAccessLogCountFromTerminal	111
UCSAPI_GetAccessLogCountFromTerminalEx	112
UCSAPI_GetAccessLogFromTerminal	114
UCSAPI_GetAccessLogFromTerminaEx	116
UCSAPI_GetAccessLogFromTerminaEx2	118

3.2.4 Authentication-related API	120
UCSAPI_SendAuthInfoToTerminal	120
UCSAPI_SendAntipassbackResultToTerminal	122
UCSAPI_SendAuthResultToTerminal	124
3.2.5 Terminal Management API.....	126
UCSAPI_GetTerminalCount	126
UCSAPI_GetFirmwareVersionFromTerminal	127
UCSAPI_UpgradeFirmwareToTerminal	128
UCSAPI_SendUserFileToTerminal	129
UCSAPI_GetOptionFromTerminal	131
UCSAPI_SetOptionToTerminal	132
UCSAPI_OpenDoorToTerminal	133
UCSAPI_OpenDoorToTerminalEx	134
UCSAPI_ControlPeripheralDevice	135
UCSAPI_SetDoorStatusToTerminal.....	137
UCSAPI_SendTerminalControl	138
UCSAPI_SetAccessControlDataToTerminal.....	139
UCSAPI_GetTerminalInfo	140
UCSAPI_SendPrivateMessageToTerminal	141
UCSAPI_SendPublicMessageToTerminal	142
UCSAPI_SendSirenToTerminal	143
UCSAPI_SetSmartCardLayoutToTerminal	145
UCSAPI_GetFpMinutiaeFromTerminal	147
UCSAPI_TerminalOptionDialog	148
UCSAPI_SetEmergencyToTerminal	149
UCSAPI_RebootToTerminal	150
3.3 Callback Event References	151
3.3.1 Events for Requests from the Terminal	151
UCSAPI_CALLBACK_EVENT_CONNECTED	151
UCSAPI_CALLBACK_EVENT_DISCONNECTED	151
UCSAPI_CALLBACK_EVENT_TERMINAL_STATUS	151
UCSAPI_CALLBACK_EVENT_ACU_STATUS	151
UCSAPI_CALLBACK_EVENT_GET_TERMINAL_TIME	151
3.3.2 Events of Response for Server Command	152
UCSAPI_CALLBACK_EVENT_REALTIME_ACCESS_LOG	152

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG	152
UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG_COUNT	152
UCSAPI_CALLBACK_EVENT_ADD_USER	152
UCSAPI_CALLBACK_EVENT_DELETE_USER	152
UCSAPI_CALLBACK_EVENT_DELETE_ALL_USERS	153
UCSAPI_CALLBACK_EVENT_GET_USER_COUNT	153
UCSAPI_CALLBACK_EVENT_GET_USER_INFO_LIST	153
UCSAPI_CALLBACK_EVENT_GET_USER_DATA	153
UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO	153
UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1	153
UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N	154
UCSAPI_CALLBACK_EVENT_VERIFY_CARD	154
UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD	154
UCSAPI_CALLBACK_EVENT_GET_TERMINAL_OPTION	154
UCSAPI_CALLBACK_EVENT_SET_TERMINAL_OPTION	154
UCSAPI_CALLBACK_EVENT_FW_UPGRADING	154
UCSAPI_CALLBACK_EVENT_FW_UPGRADED	155
UCSAPI_CALLBACK_EVENT_FW_VERSION	155
UCSAPI_CALLBACK_EVENT_USERFILE_UPGRADING	155
UCSAPI_CALLBACK_EVENT_USERFILE_UPGRADED	155
UCSAPI_CALLBACK_EVENT_OPEN_DOOR	155
UCSAPI_CALLBACK_EVENT_TERMINAL_CONTROL	155
UCSAPI_CALLBACK_EVENT_PICTURE_LOG	155
UCSAPI_CALLBACK_EVENT_ANTIPASSBACK	156
UCSAPI_CALLBACK_EVENT_SET_ACCESS_CONTROL_DATA	156
UCSAPI_CALLBACK_EVENT_REGISTER_FACE	156
UCSAPI_CALLBACK_EVENT_GET_ACU_OPTION	157
UCSAPI_CALLBACK_EVENT_SET_ACU_OPTION	157
UCSAPI_CALLBACK_EVENT_GET_ACU_LOCKSCHEDULE	157
UCSAPI_CALLBACK_EVENT_SET_ACU_LOCKSCHEDULE	157
UCSAPI_CALLBACK_EVENT_GET_SIREN	158
UCSAPI_CALLBACK_EVENT_SET_SIREN	158
UCSAPI_CALLBACK_EVENT_SET_SMARTCARD_LAYOUT	158
UCSAPI_CALLBACK_EVENT_FP_MINUTIAE	158
3.4 Error definitions.....	159

3.4.1 Success	159
UCSAPIERR_NONE	159
3.4.2 General error definitions	159
UCSAPIERR_INVALID_POINTER	159
UCSAPIERR_INVALID_TYPE	159
UCSAPIERR_INVALID_PARAMETER	160
UCSAPIERR_INVALID_DATA	160
UCSAPIERR_FUNCTION_FAIL	160
UCSAPIERR_NOT_SERVER_ACTIVE	160
UCSAPIERR_INVALID_TERMINAL	160
UCSAPIERR_PROCESS_FAIL	161
UCSAPIERR_USER_CANCEL	161
UCSAPIERR_UNKNOWN_REASON	161
3.4.3 Data size related error definitions	161
UCSAPIERR_CODE_SIZE	161
UCSAPIERR_USER_ID_SIZE	162
UCSAPIERR_USER_NAME_SIZE	162
UCSAPIERR_UNIQUE_ID_SIZE	162
UCSAPIERR_INVALID_SECURITY_LEVEL	162
UCSAPIERR_PASSWORD_SIZE	162
UCSAPIERR_PICTURE_SIZE	163
UCSAPIERR_INVALID_PICTURE_TYPE	163
UCSAPIERR_RFID_SIZE	163
UCSAPIERR_MAX_CARD_NUMBER	163
UCSAPIERR_MAX_FINGER_NUMBER	163
3.4.4 Authentication-related error definitions	164
UCSAPIERR_INVALID_USER	164
UCSAPIERR_UNAUTHORIZED	164
UCSAPIERR_PERMISSION	164
UCSAPIERR_FINGER_CAPTURE_FAIL	164
UCSAPIERR_DUP_AUTHENTICATION	165
UCSAPIERR_ANTIPASSBACK	165
UCSAPIERR_NETWORK	165
UCSAPIERR_SERVER_BUSY	165
UCSAPIERR_FACE_DETECTION	165

4. API Reference for COM	167
4.1 UCSAPI Object	167
4.1.1 Properties	167
ErrorCode	167
EventError	167
ConnectionsOfTerminal	168
TerminalUserData	168
ServerUserData	168
AccessLogData	168
AccessControlData	169
TerminalMacAddr	169
SampleNumber	169
TotalFingerCount	169
InitPacketCrypto	170
InitAlpetaSync	170
TerminalAuthTypeIris	170
TerminalAuthTypePassword	170
TerminalAuthTypeFace	171
TerminalAuthTypeCard	171
TerminalAuthTypeFinger	171
Terminal Authentication Type WT Face.....	171
4.1.2 Methods	172
ServerStart.....	172
ServerStop	173
SetTerminalTime	174
SetTerminalTimezone	175
SetError	177
GetTerminalCount	178
GetFirmwareVersionFromTerminal	179
UpgradeFirmwareToTerminal	180
SendUserFileToTerminal	181
OpenDoorToTerminal.....	182
SetDoorStatusToTerminal	182
SendTerminalControl	183
SendPrivateMessageToTerminal	184

SendPublicMessageToTerminal	185
SetWiegandFormatToTerminal	187
SetDoorStatusToACU	188
GetFpMinutiaeFromTerminal	188
SetEmergencyToTerminal	189
TerminalOptionDialog	190
OpenDoorToTerminalEx	191
GetUserInfoListFromTerminal	192
ControlPeripheralDevice	192
get_FingerID	193
get_FPSampleData.....	194
EnrollFromTerminal	194
get_FPSampleDataLength	195
Reboot to Terminal	196
4.2 IServerUserData Interface	197
4.2.1 Properties	197
UserID	197
UniqueId	197
UserName	197
AccessGroup	198
SecurityLevel	198
IsCheckSimilarFinger	198
IsAdmin	199
IsIdentify	199
Password	199
FaceNumber	199
FaceData	200
IsFace1toN.....	200
IsBlacklist	200
IsIris1toN	200
4.2.2 Methods	201
InitUserData	201
SetAuthType	201
SetFPSampleData	202
AddFingerData	203

SetDuressFinger	204
SetCardData	204
SetPictureData	206
SetAccessDate	207
SetAccessTime	208
AddUserToTerminal.....	209
SetAuthTypeEx.....	210
SetWalkThroughData	211
SetIrisData	211
Reset Duress Finger	212
4.3 ITerminalUserData Interface	214
4.3.1 Properties	214
CurrentIndex / TotalNumber	214
UserID	214
UniqueID	215
UserName	215
AccessGroup	215
IsAdmin	216
IsIdentify	216
Access Date Type	216
Start Access Date/End Access Date	217
StartAccessTime/EndAccessTime	217
SecurityLevel	218
IsAndOperation	218
IsFinger	219
IsFPCard	219
IsCard	220
IsCardID.....	220
IsPassword	221
Password	221
CardNumber.....	221
RFID	222
PictureDataLength	222
PictureData	223
TotalFingerCount	223

FingerID	224
SampleNumber	224
FPSampleData	224
FaceNumber	225
FaceData	225
IsBlacklist	226
IsFace	226
Isliris	227
IsMobileKey	227
WalkThroughData	227
WalkThroughLength	228
Walkthrough Type	228
WalkThroughData_2	229
WalkThroughLength_2	229
WalkThroughType_2	229
IrisDataLength	230
IrisData	230
MaxUserNumber	231
RegFingerNumber	231
MaxFingerNumber	231
RegFaceNumber	232
MaxFaceNumber	232
4.3.2 Methods	234
GetUserCountFromTerminal	234
GetUserDataFromTerminal	236
DeleteUserFromTerminal	238
DeleteAllUsersFromTerminal	239
RegistFaceFromTerminal	239
get_FPSampleDataLength	240
Register Iris From Terminal	241
Register Walk-Through Face from Terminal	242
GetUserInfoListFromTerminal	243
4.4 IAccessLogData Interface	245
4.4.1 Properties	245
CurrentIndex / TotalNumber	245

UserID	245
AuthType	246
AuthMode	246
DateTime	246
IsAuthorized	247
RFID	247
PictureDataLength	247
PictureData	248
FingerImageData	248
FingerImageLength	249
FingerImageFormat	249
Latitude	250
Longitude	250
IsDrinking	250
Drinking Level	251
ThermalBurnType	251
ThermalBurn	252
4.4.2 Methods	253
SetPeriod	253
GetAccessLogCountFromTerminal	253
GetAccessLogFromTerminal	255
SetFingerImageSend	256
GetAccessLogFromTerminalEx2	256
4.5 IAccessControlData Interface	258
4.5.1 Properties	258
4.5.2 Methods	259
InitData	259
SetTimeZone	260
SetAccessTime	262
SetHoliday	263
SetAccessGroup	264
Set Access Control Data to Terminal	265
SetTimeZoneToAuthProperty	266
SetAuthProperty	267
4.6 IServerAuthentication Interface	269

4.6.1 Properties	269
DeviceID	269
4.6.2 Methods	270
SetAuthType	270
SendAuthInfoToTerminal	272
SendAuthResultToTerminal	273
SendAntipassbackResultToTerminal	275
SetAuthTypeEx.....	276
4.7 ITerminalOption Interface	277
4.7.1 Properties	277
flagSecuLevel / flagInputIDLength / flagAutoEnterKey / flagSound / flagAuthentication /	
flagApplication / flagAntipassback / flagNetwork / flagInputIDType / flagAccessLevel /	
flagPrintText / flagSchedule	277
SecurityLevel_1To1 / SecurityLevel_1ToN	277
InputIDLength	278
AutoEnterKey.....	279
Sound	279
Authentication	279
Application	280
Antipassback	280
NetworkType / TerminalIP / Subnet / Gateway / ServerIP / Port	281
InputIDType	282
AccessLevel	282
PrintText	283
IsUse / StartHour / StartMinute / EndHour / EndMinute	284
Month / Day	285
HolidayType	285
Port	286
ServerIP	286
NetworkType	286
flagSchedule	287
flagPrintText	287
flagAccessLevel	287
flagInputIDType	288
flagServer	288

flagNetwork.....	288
flagAntipassback	289
flagApplication	289
flagAuthentication	289
flagSound	290
flagAutoEnterKey	290
flagInputDLength	290
flagSecuLevel.....	291
4.7.2 Methods	292
SetOptionToTerminal	292
GetOptionFromTerminal	293
SetDaySchedule.....	295
GetDaySchedule	297
SetHoliday	298
GetHoliday	299
Clear	300
get_ACUStatusValue	301
ACUGetReaderVersion	301
GetOptionFromACU	302
SetOptionToACU	303
GetLockScheduleFromACU	304
SetLockScheduleToACU	304
ClearSirenConfig.....	305
SetSirenConfig	305
SetSirenToTerminal	305
GetSirenFromTerminal	306
GetSirenConfig	306
4.8 ISmartCardLayout Interface	307
4.8.1 Properties	307
SectorNumber	307
CardType	307
ReadType	308
SerialFormat	308
4.8.2 Methods	308
ClearSectorLayout	308

SetSectorLayout	309
SetSmartCardLayoutToTerminal	310
4.9 Events of COM	311
EventUserFileUpgrading	311
EventUserFileUpgraded	312
EventRegistFace	312
Event ACU Status	313
EventGetLockScheduleFromACU	314
EventSetLockScheduleToACU	314
EventSetSirenToTerminal	315
Get Siren from Terminal	315
EventSetSmartCardLayout	315
EventGetTerminalTime	316
EventGetFpMinutiaeFromTerminal	316
5. Error definitions	317
5.1 Success	317
UCSAPIERR_NONE	317
5.2 General error definitions	317
UCSAPIERR_INVALID_POINTER	317
UCSAPIERR_INVALID_TYPE	317
UCSAPIERR_INVALID_PARAMETER	318
UCSAPIERR_INVALID_DATA	318
UCSAPIERR_FUNCTION_FAIL	318
UCSAPIERR_NOT_SERVER_ACTIVE	318
UCSAPIERR_INVALID_TERMINAL	318
UCSAPIERR_PROCESS_FAIL	319
UCSAPIERR_USER_CANCEL	319
UCSAPIERR_UNKNOWN_REASON	319
5.3 Data size-related error definitions	319
UCSAPIERR_CODE_SIZE	319
UCSAPIERR_USER_ID_SIZE	319
UCSAPIERR_USER_NAME_SIZE	320
UCSAPIERR_UNIQUE_ID_SIZE	320
UCSAPIERR_INVALID_SECURITY_LEVEL	320
UCSAPIERR_PASSWORD_SIZE	320

UCSAPIERR_PICTURE_SIZE	320
UCSAPIERR_INVALID_PICTURE_TYPE	321
UCSAPIERR_RFID_SIZE	321
UCSAPIERR_MAX_CARD_NUMBER	321
UCSAPIERR_MAX_FINGER_NUMBER	321
5.4 Authentication-related error definitions	321
UCSAPIERR_INVALID_USER	322
UCSAPIERR_UNAUTHORIZED	322
UCSAPIERR_PERMISSION	322
UCSAPIERR_FINGER_CAPTURE_FAIL	322
UCSAPIERR_DUP_AUTHENTICATION	322
UCSAPIERR_ANTIPASSBACK	323
UCSAPIERR_NETWORK	323
UCSAPIERR_SERVER_BUSY	323
UCSAPIERR_FACE_DETECTION	323
UCSAPIERR_BLACKLIST	323

1. Overview

The UCS (UNION COMMUNITY Server) SDK is compatible with Union Community Co., Ltd.'s network-based fingerprint recognition terminals.

It was developed as a High Level SDK to facilitate application development.

The UCS SDK provides the interfaces (Application Programming Interface, API) necessary for developing fingerprint recognition server applications.

This is provided to offer an API and can be used with the UCBioBSP SDK for fingerprint registration and authentication.

1.1 Application

The UCS SDK is a server application that can be integrated with network-based terminal products provided by Union Community Co., Ltd.

Defining the Server Application Programming Interface (API). Therefore, access control, attendance

When developing applications for tasks such as attendance, student records, and academic management, applying this approach enables the development of programs that are easier to use and more stable. can be utilized.

1.2 Features

■ Centralized Management Approach

By having terminals connect to the UCS SDK, all terminals can be centrally managed from a single point.

One approach offers significant advantages in network configurations utilizing aerial networks.

If you use a public network that employs a metered system (a system that charges usage fees based on the amount of time the line is used),

We will use a method that connects directly to the terminal for management without executing the server functions of the SDK we are using.

It is possible.

■ Provision of various APIs for device management

Provides various APIs for user management, log management, access control management, and more.

■ Provides various development modules and sample source code

The UCS SDK is designed not only for C/C++ developers but also for developers using Visual Basic, Delphi, DotNet, and other languages.

To facilitate development in these tools, COM-based modules and DotNet class libraries

It is provided together. Additionally, sample source code required for SDK usage is provided together.

- Provides various authentication methods

Means of user identification include passwords, cards, and various combinations of these authentication methods in addition to fingerprints.

Provides.

1.3 Development Environment

All modules provided by the UCS SDK were compiled using VC++ 6.0 and are compatible with most 32-bit compilers such as Visual C++.

Programming is possible using this SDK in the compiler.

Additionally, for developers using Visual Basic, Delphi, DotNet, and other languages, not just C/C++ developers, this

It provides both COM-based modules and DotNet class libraries to facilitate development in the tools.

Referring to the respective sample sources will be very helpful for understanding how to use the modules.

1.4 Module Configuration

- Basic Module: UCSAPI40.dll

As the core module of the UCS SDK, it is the main module that implements all functions related to communication with the terminal.

It is possible. When developing using the UCS SDK, this module must be included.

It provides APIs that can be used in C and C++.

The related sample code can be found in the DLL folder within the provided Samples folder.

- COM Module: UCSAPICOM.dll

Supports users of RAD (Rapid Application Development) tools such as Visual Basic and Delphi, as well as DotNet.

It is a COM (Component Object Model) module developed for the purpose of.

UCSAPICOM.dll exists at a higher level than UCSAPI40.dll, so UCSAPI40.dll must also be present.

It must be enabled to function. Furthermore, while it does not provide all the features offered by UCSAPI40.dll, conversely,

It also possesses some features not provided by UCSAPI40.dll.

The related sample code can be found in the COM folder within the provided Samples folder.

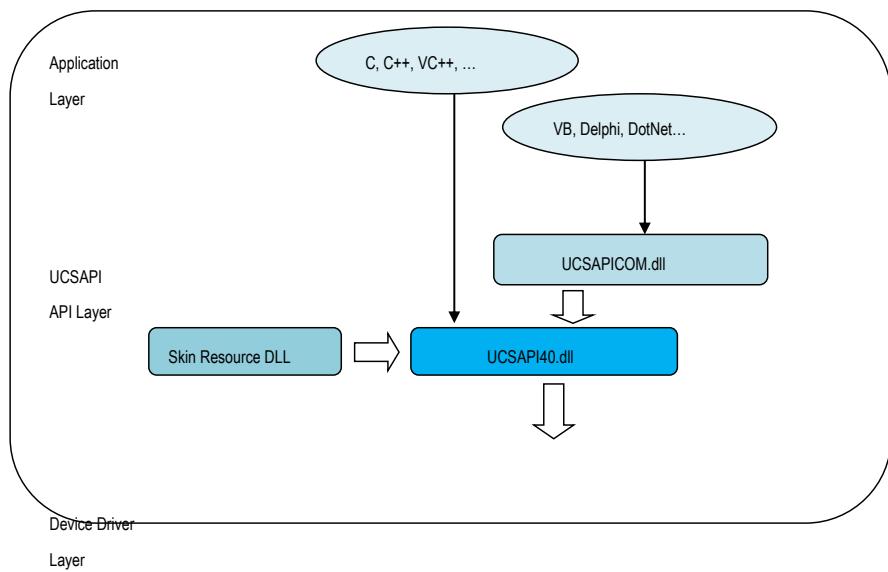
- Winsock Engine Module: WSEngine.dll

This module handles Socket IO (Input/Output). WSEngine.dll operates at a lower level than UCSAPI40.dll.

Repeats.

1.5 Development Model

The structure of the development model is as follows.



1.6 Glossary

■ Terminal

This refers to the network-based fingerprint recognition terminal provided by Union Community Co., Ltd.

■ Terminal ID

A key value used to distinguish terminals connected to a single system, assigning a unique ID to each terminal.

Refer to the manual for the specific device to learn how to assign it.

■ Client

This refers to the application that communicates with the provided network-based fingerprint recognition terminal.

■ Client ID (ClientID)

A client ID is used when developing applications using a client/server model.

Server modules require a key to distinguish each client in order to support multi-client environments.

This is referred to as the client ID.

■ 1:1 Verification

To verify an individual's identity, the fingerprint template (or card, password) corresponding to the user ID and the submitted sample

This is the one-to-one processing procedure for comparing files.

■ 1:N Authentication (1 to N, Identification)

One-to-many matching that compares some or all fingerprint templates with submitted samples to verify an individual's identity.

It is a processing step.

■ Authentication Type

Defines the authentication type used during user authentication.

The user enters their UserID or UniqueID from the terminal for authentication, and the fingerprint recognition terminal authenticates with the server.

When using this method, the terminal authenticates with the server to obtain the authentication type of the user registered on the server.

You will be prompted to select a type.

Type	Value	Contents
1:N Fingerprint	0 1:N Fingerprint Authentication	

1:1 Fingerprint	1 1:1 Fingerprint Authentication
Card & Fingerprint	2. Store fingerprint information on the smart card to input the fingerprint and Perform 1:1 authentication between stored fingerprints
Card	Three-Factor Authentication
Password	4 Password Authentication

■ Registration Authentication Type

Defines the authentication types available during user registration.

Means of user identification include passwords, cards, and various combinations of these authentication methods in addition to fingerprints.

Provides.

Type	Contents
Fingerprint	Fingerprints are used as an authentication method.
Fingerprint Card	A fingerprint card is used as an authentication method. Fingerprint cards store an individual's fingerprint template on a smart card for authentication. It performs a 1:1 authentication between the input sample and the stored template.
Password	Passwords are used as a means of authentication. The password is a string value of up to 8 characters.
Card	Use a card as an authentication method.
Card or Fingerprint	Use a card or fingerprint as an authentication method.
Card and Fingerprint	A combination of card and fingerprint is used as an authentication method.
Card or Password	Use a card or password as an authentication method.
Card and Password	It uses a combination of a card and fingerprint as an authentication method.
(ID and Fingerprint) or (Card and Fingerprint)	As an authentication method, a combination of ID and fingerprint or a combination of card and fingerprint is used. Use.
(ID and Password) or (Card and Password)	As an authentication method, a combination of ID and password or a card and password Use the sum.
Fingerprint and Password	It uses a combination of fingerprint and password as an authentication method.
Fingerprint or Password	After fingerprint authentication fails, the password is used as the authentication method.
Card and Password and Fingerprint	It uses a combination of card, password, and fingerprint as authentication methods.

■ Fingerprint Authentication Level (Security Level)

Defines the security level to use during fingerprint authentication, with values ranging from 1 to 9. The UCS SDK provides developers with

The following level values are recommended. Higher level values result in a higher false rejection rate (FRR).

The false acceptance rate (FAR) decreases. The UCS SDK supports Level 4 for Verification (1:1) authentication and Identification (1:N) authentication.

We recommend Level 5 as the baseline level. Applications should set the authentication level based on the baseline level when the FAR is high.

It can be adjusted upward, and the higher the FRR, the more the authentication level can be adjusted downward.

■ Terminal Authentication Mode

Defines the behavior for user authentication and log recording.

Mode	Value	Content
N/S	0	User authentication is performed on the server during online operation, and on the terminal during offline operation. Perform user authentication. During server authentication, the server stores authentication records. and when the network is disconnected, the terminal stores the authentication records and when the server connection is reestablished, Transmits the authentication records stored on the terminal to the server.
Serial Number	1.	User authentication is performed on the terminal during online operation. However, it is stored on the terminal. For user authentication requests that have not been made, send an authentication request to the server. Authentication records are always stored on the terminal device, and only the authentication results are sent to the server during online sessions. and store it by sending it.
N/O		User authentication is performed only on Server 2. User authentication is performed offline. I can't.
S/O		User authentication is performed only on the terminal. During online sessions, only authentication logs are sent to the server. sends to.
S/S	4	The terminal does not attempt to connect to the server; it merely connects to the application. Waiting. The terminal performs authentication.

■ Terminal Application Mode

Defines the program operation mode supported by the terminal.

Mode	Value	Content
Access Control	0	The terminal is operated for access control purposes.
Attendance	1	The terminal is operated for attendance management purposes.
Drinking water	2	The terminal is operated for the purpose of water management.

■ User Authentication Mode

Define the purpose of authentication during user authentication. To use extended modes beyond the default authentication mode.

If so, you can use it by setting the terminal's extended mode option. The authentication mode allows the program to be used for attendance and

It can be used for drinking water purposes.

Mode	Value	Content
Commuting to work	0	Authenticate in work mode.
Leaving work	1.	Verify you're in off-duty mode.
General	2.	Authenticate in normal mode.
Field work	3.	Authenticate in off-site mode.
Return	4.	Authenticate in return mode.

■ Log Acquisition Type

The UCS SDK defines three types of log categories to retrieve log data from the terminal.

The number of logs that can be stored internally varies depending on the terminal model, and the maximum storage capacity is measured in seconds.

Some records prior to the display will be deleted to store new logs.

Type	Value	Content
New Log	New logs	not sent to the server
Old Log	Logs already	successfully transmitted to the server
All Logs	2. All logs	stored on the device

■ User Property

A 1-byte data field that defines whether the user is an administrator and the authentication type.

Each field can be assigned a value of 1 or 0. To use the property value of that field, assign a value of 1.

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	Operation (AND) or (OR)	Card ID	Card	Password	Reserved	Fingerprint

-Admin: You can designate a user as a device administrator.

-Identify: Users can be designated for 1:N fingerprint authentication.

-Operation: You can specify that each authentication type be combined using AND or OR operations.

Here, set 1 for an AND combination and 0 for an OR combination.

-CardID: You can designate the RFID to be used as a UserID or UniqueID. The CardID is the RFID of the card.

This refers to using it solely as an identifier, like a UserID, rather than as an authentication method.

It must be specified in combination with other authentication types using an AND operator.

-Card: You can designate users as eligible for Card authentication.

-Password: You can specify whether the user can be authenticated by password.

-Fingerprint: You can designate users to be authenticated via fingerprint.

User attributes can be expressed using the following 12 values.

① Fingerprint

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	0	0	0	0	1

② Fingerprint Card

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	0	0	0	1	0

③ Password

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	0	0	1	0	0

④ Card

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	0	1	0	0	0

⑤ Card or Fingerprint

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	0	1	0	0	1

⑥ Card and Fingerprint

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	1	0	1	0	0	1

⑦ Card or Password

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	0	1	1	0	0

⑧ Card and Password

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	1	0	1	1	0	0

⑨ (ID and Fingerprint) or (Card and Fingerprint)

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	1	0	0	0	1

⑩ (ID and Password) or (Card and Password)

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	1	0	1	0	0

⑪ Fingerprint and Password

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	1	0	0	1	0	1

⑫ Fingerprint or Password: Password authentication when fingerprint authentication fails

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	Identify	0	0	0	1	0	1

⑬ Card, Password, and Fingerprint

7Bit	6Bit	5Bit	4Bit	3Bit	2Bit	1Bit	0Bit
Admin	0	1	0	1	1	0	1

■ Terminal Admin

The terminal administrator is a user with the authority to modify the terminal's configuration settings or register/delete users.

If one or more device administrators are saved on the device, enter the device's settings screen.

The city administrator login process is required. To ensure secure device usage, the device administrator must be registered.

Therefore, we recommend using it.

■ Device Status

The terminal periodically or upon status changes transmits its own status information and the status information of devices connected to it.

Sends status information to the server.

Device lock status:

This value indicates the device's operable state.

The SDK can set the device's operation state to Lock/Unlock. When in Lock state, the device's

The connection to Ro Geon and the network is maintained, but operation and access to terminals other than the administrator's are impossible.

Once a terminal enters the Lock state, even keypad operation becomes impossible.

Lock Control Status:

This value indicates the status of the lock device connected to the terminal.

The SDK can set/release the status of the lock device connected to the terminal to the open state, and open

In this state, access becomes possible without user authentication.

Lock monitoring status:

This value outputs the open/closed status value received from the monitored locking device.

Terminal cover status:

This value indicates the open/closed status of the terminal cover.

Status	Value	Content
Device Status	0Normal	
	1 Locked	state
Lock Control Status	0 Closed	state
	1 Open	state
Lock Monitoring Status	0 Closed	state
	1 Open	state
	2. Without monitoring	
Terminal cover open	0 Closed	state
	1 Open	state

< Device Status Information Summary >

2. Installation

2.1 System Requirements

- CPU

Intel Pentium 133MHz or higher

- Memory

16M or more

- USB Port

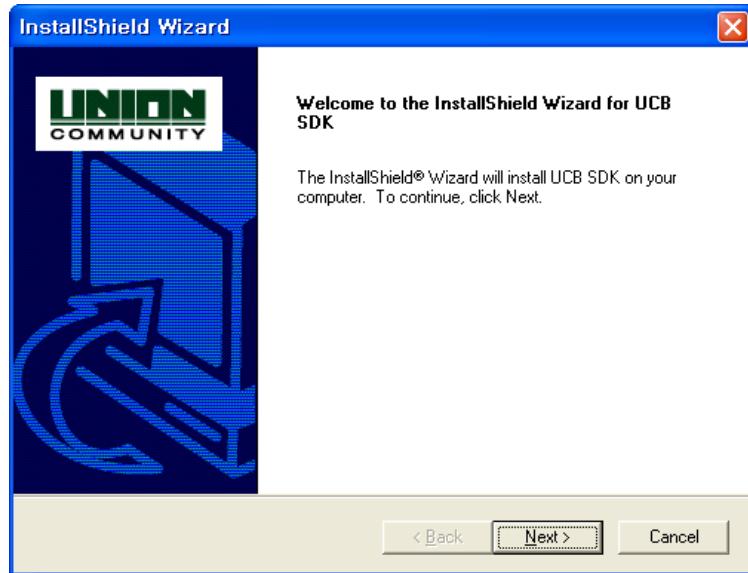
USB 1.1

- OS

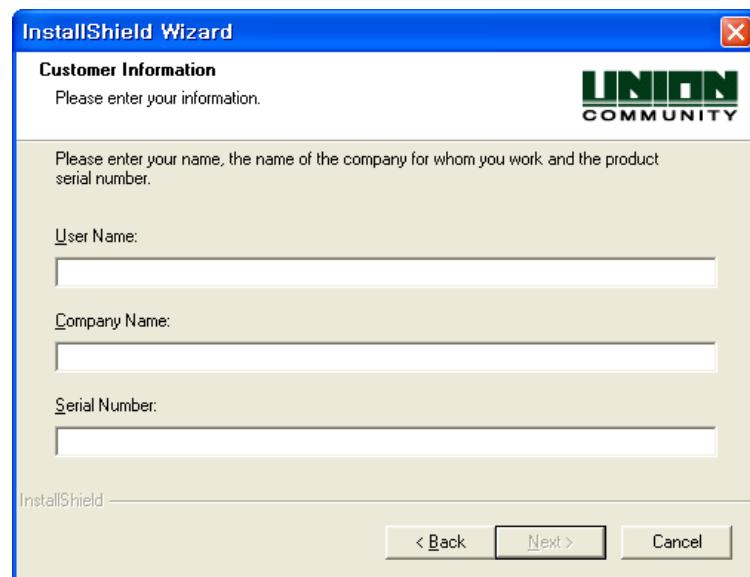
Windows 98/ME or 2000/XP/2003/Vista/Windows 7

2.2 Installation

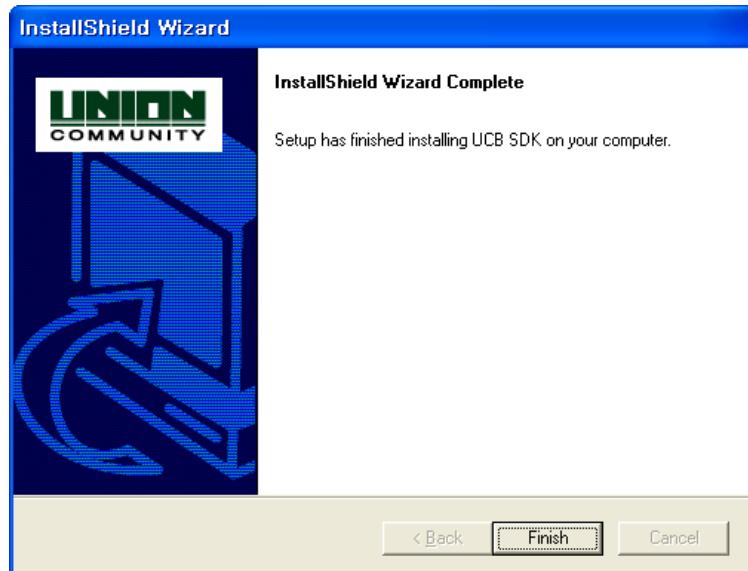
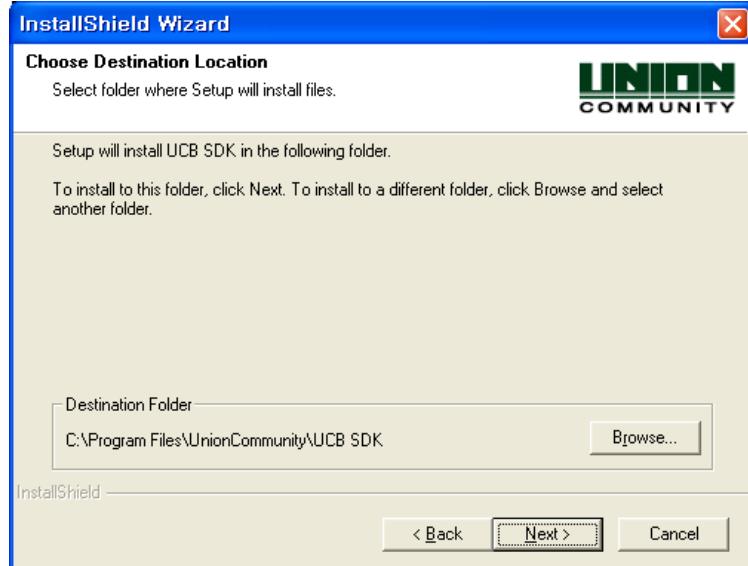
When you insert the installation CD, Setup.exe will run automatically.



Please install while checking the contents step by step.



Enter your user information and the product's serial number.



2.3 Files to be Installed

Once the SDK installation is fully complete, the following files will be installed in the specified installation folder.

2.3.1 Windows System Directory

The Core module for using the SDK is installed. The following files are installed.

UCSAPI40.dll

Core module of the UCS SDK. Responsible for executing all SDK functions.

UCSAPICOM.dll

COM Module for RAD Tool Developers.

WSEngine.dll

Communication Module for Windows Socket I/O Processing

2.3.2 GAC (Global Assembly Cache) Folder

The following files are installed in the GAC folder where the Class Library for the .NET Framework environment is installed.

When installing the SDK, the .NET library is installed if it is included in the installation.

2.3.3 (Installation Folder)\Bin

It contains the core files required for SDK execution and executable files for samples.

UCSAPI40.dll / UCSAPICOM.dll / WSEngine.dll

The same file installed in the Windows system32 folder.

Demo application

The UCS SDK includes numerous demo programs that allow you to quickly test its features.

All demo program source files are provided in the Samples folder.

2.3.4 (Installation Folder)\dotNET

It contains the dotNET Class Library files required for SDK execution.

UNIONCOMM.SDK.UCSAPI40.dll

Class Library module for .NET. The same file installed in the GAC.

2.3.5 (Installation Folder)\dotNET\Setup

It contains an installation file for installing the Class Library for .NET into the GAC.

Setup.exe (UCSAPI40.NET_Setup.msi)

Class Library Installation File for .NET

2.3.6 (Installation Folder)\Inc

UCSAPI.h

The main header file for the UCS SDK. Including this file internally includes UCSAPI_Basic.h,

The UCSAPI_Error.h and UCSAPI_Type.h files are automatically included.

UCSAPI_Basic.h

Defines the basic data types used in the UCS SDK.

UCSAPI_Error.h

Defines the error values used in the UCSAPI40 module.

UCSAPI_Type.h

It defines the data types and structure information used in the UCS SDK.

2.3.7 (Installation Folder)\Lib

It contains the Link library file for development in VC++ using the SDK.

UCSAPI40.lib

Library file for Linking with VC++. When statically linking UCBioBSP.dll in VC++, the following applies:

It is used.

2.3.8 (Installation Folder) \Samples

Sample source code for each language is organized into separate folders.

DLL

It contains sample code that can be developed using UCSAPI40.dll.

1) VC6: It contains samples created for Visual C++ 6.0.

COM

It contains sample code that can be developed using UCSAPICOM.dll.

1) VB6: It contains samples created for Visual Basic 6.0.

dotNET

Sample Developable in Microsoft's .NET Environment Using UNIONCOMM.SDK.UCSAPI40dll

It contains code.

1) C#: Visual Studio.NET 2005 includes samples created for C#.

2.3.9 (Installation Folder)\Skins

Skin resource files for each language are included. Currently, only English and Korean are included.

3. API Reference for DLL

This chapter describes the types and APIs for using the DLL module UCSAPI40.dll.

3.1 Type Definitions

3.1.1 Basic types

Defined in UCSAPI_Basic, it specifies the basic types. For OS- and CPU-independent development.

We are redefining the base types. The following explanation assumes development in C++ on a standard Windows environment.

It is explained based on the standard.

UCSAPI_SINT8 / UCSAPI_SINT16 / UCBioAPI_SINT32

Signed 1-byte / 2-byte / 4-byte values

UCSAPI_UINT8 / UCSAPI_UINT16 / UCBioAPI_UINT32

Unsigned 1-byte / 2-byte / 4-byte values

UCSAPI_SINT / UCSAPI_UINT

The value of int / unsigned int varies depending on the OS. On a 32-bit OS, it operates as 4 bytes, while on a 64-bit OS, it operates as

It operates with 8 bytes.

UCSAPI_VOID_PTR

Void*

UCSAPI_BOOL

UCSAPI_FALSE(0) / UCBioAPI_TRUE(1) values are possible. Handled identically to int.

UCSAPI_CHAR / UCSAPI_CHAR_PTR

Char and char* refer to 1-byte character and string values.

UCSAPI_NULL
Represents NULL. Defined as the value of <void*>0).

UCSAPI_HWND
HWND value, meaning a Windows handle.

3.1.2 General Types

It is declared in UCSAPI_Type.h and defines common types.

UCSAPI_RETURN
Prototype:
`typedef UCSAPI_UINT32 UCSAPI_RETURN;`

Description:
Defines the values returned by UCSAPI SDK functions. Typically holds error values from the UCS SDK.
For specific error values, refer to the ERROR definition.

UCSAPI_DATE_TIME_INFO
Prototype:
`typedef struct ucsapi_datetime_info`
{
 UCSAPI_UINT16 Year;
 UCSAPI_UINT8 Month;
 UCSAPI_UINT8 Day;
 UCSAPI_UINT8 Hour;
 UCSAPI_UINT8 Min;
 UCSAPI_UINT8 Sec;
 UCSAPI_UINT8 Reserved;
};
`UCSAPI_DATE_TIME_INFO, *UCSAPI_DATE_TIME_INFO_PTR;`

Description:
A structure containing date and time information.

UCSAPI_MESSAGE

```
#define UCSAPI_MESSAGE
```

128

3.1.3 User Information Related Types

It is declared in UCAPIL_Type.h and defines types related to user information.

UCSAPI_ACCESS_DATE_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_ACCESS_DATE_TYPE;
```

Description:

Defines the data types for the UCSAPI_ACCESS_DATE structure. Access period data is not used, access type _____

A total of three data types can be specified, such as the period of operation and the period of inaccessibility. Possible values

The following is as follows.

#define UCSAPI_DATE_TYPE_NOT_USE	0
#define UCSAPI_DATE_TYPE_ALLOW	1
#define UCSAPI_DATE_TYPE_RESTRICTION	2

UCSAPI_ACCESS_AUTHORITY

Prototype:

```
typedef struct ucsapi_access_authority
{
    UCSAPI_DATA_PTR           AccessGroup;
    UCSAPI_ACCESS_DATE_TYPE   Access Date Type;
    UCSAPI_ACCESS_DATE_PTR    AccessDate;
    UCSAPI_TIME_HH_MM_PTR     AccessStartTime;
    UCSAPI_TIME_HH_MM_PTR     AccessEndTime;
} UCSAPI_ACCESS_AUTHORITY, UCSAPI_ACCESS_AUTHORITY_PTR;
```

Includes note[w1]: Added on 20250418

Description:

A structure containing the user's access permission information. The description of each value is as follows.

AccessGroup:

A pointer to a structure containing access permission group code information.

Access Date Type:

Specifies the data type held by the UCSAPI_ACCESS_DATE structure.

AccessDate:

A pointer to a structure containing access period information.

AccessStartTime:

A pointer to a structure containing entry and exit time information.

AccessEndTime:

A pointer to a structure containing entry and exit time information.

UCSAPI_CARD_DATA

Prototype:

```
typedef struct ucsapi_card_data
{
    UCSAPI_UINT32           CardNumber;
    UCSAPI_DATA_PTR          RFID[UCSAPI_CARD_NUMBER_MAX];
} UCSAPI_CARD_DATA, UCSAPI_CARD_DATA_PTR;
```

Description:

A structure containing card information. The description of each value is as follows.

CardNumber:

Specify the total number of RFID tags. The array contains RFID information for the specified number of tags.

RFID:

Pointer array for the structure containing RFID information.

UCSAPI_FINGER_DATA

Prototype:

```
typedef struct ucsapi_finger_data
{
```

```

UCSAPI_UINT32          SecurityLevel;
UCSAPI_UINT8           TemplateFormat;
UCSAPI_UINT8           DuressFinger[10];
UCSAPI_BOOL            IsCheckSimilarFinger;
UCSAPI_EXPORT_DATA_PTR ExportData;

UCSAPI_FINGER_DATA, UCSAPI_FINGER_DATA_PTR;

```

Description:

A structure containing fingerprint information. The description of each value is as follows.

SecurityLevel:

Specify the security level to use during authentication.

The possible values are defined in the UCBioAPI_FIR_SECURITY_LEVEL definition of the UCBioBSP SDK.

TemplateFormat:

Specify the feature point type. Depending on the feature point type, the possible values are:

```

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

```

This is one of these values, and the default is 0. (See UCBioAPI_Type.h)

DuressFinger:

Composed of 10 bytes, each byte value indicates the presence or absence of a blackmail fingerprint (0: normal, 1: blackmail fingerprint)

The position of each byte value is indicated by the finger ID, which is one less than the Finger ID.

When the threat fingerprint is entered, the terminal processes it as a successful authentication and passes the result value set to 33.

IsCheckSimilarFinger:

Specifies whether to check for similar fingerprints when adding user fingerprint data to the device.

If this value is set to True, the terminal compares the fingerprint with those of all registered users to detect a similar fingerprint.

If a duplicate fingerprint is detected during the existence check, the registration is treated as a failure. This flag is sent to the terminal as the user's

Slowing down additional operations becomes a factor that degrades performance when there are many fingerprint registration users.

Used when calling UCSAPI_AddUserToTerminal.

ExportData:

A pointer to the structure that will hold the converted Template data.

UCBioBSP SDK's UCBioAPI_EXPORT_DATA Structure Reference.

UCSAPI_FACE_INFO

Prototype:

```
typedef struct ucsapi_face_info
{
    Long Length;
    BYTE* Data;
    UCSAPI_FACE_INFO, *UCSAPI_FACE_INFO_PTR;
```

Description:

A structure containing facial feature point data extracted from a single photograph.

length:

Has the size of Data

Data:

Face feature point data extracted from an actual input face image; typically, in the case of the first image, it is extracted.

The initial data is approximately 20K, and subsequent data is approximately 4K.

UCSAPI_FACE_DATA

Prototype:

```
typedef struct ucsapi_face_data
{
    long FaceNumber;
    UCSAPI_FACE_INFO_PTR FaceInfo[UCSAPI_MAX_FACE_NUMBER];
    UCSAPI_FACE_DATA, *UCSAPI_FACE_DATA_PTR;
```

Description:

A single user can have up to 10 facial feature point data entries.

FaceNumber:

The number of registered face data points represents the valid count of subsequent Face Info entries.

FaceInfo:

The actual facial feature point information entered during registration: 5 points for standard registration and 3 points for simplified registration per session.

Since registration is possible twice, FaceInfo can have a minimum of 3 and a maximum of 10 entries.

UCSAPI_IRIS_INFO

Prototype:

```
typedef struct ucsapi_iris_info
{
    UCSAPI_UINT32 Length;
    UCSAPI_UINT8* Data;
    UCSAPI_IRIS_INFO, *UCSAPI_IRIS_INFO_PTR;
```

Description:
A structure containing iris feature point data extracted from an image.

Length:
Has the size of Data

Data:
Iris feature point data extracted from the actual iris image input

UCSAPI_AUTH_DATA

Prototype:

```
typedef struct ucsapi_auth_data
{
    UCSAPI_DATA_PTR           Password;
    UCSAPI_CARD_DATA_POINTER  Card;
    UCSAPI_FINGER_DATA_PTR   Finger;
    UCSAPI_FACE_DATA_PTR     Face;

    UCSAPI_AUTH_DATA, UCSAPI_AUTH_DATA_PTR;
```

Description:
A structure containing authentication information. The description of each value is as follows.

Password:
A pointer to a structure containing password information.

Card:
A pointer to a structure containing card information.

Finger:
A pointer to a structure containing fingerprint information.

Face:
A pointer to a structure containing facial information.

UCSAPI_PICTURE_HEADER

Prototype:

```
typedef struct ucsapi_picture_header
{
    UCSAPI_UINT8          Format[4];      /* must be "jpg" */
    UCSAPI_UINT32         Length;        /* Maximum length is 7 kilobytes */

    UCSAPI_PICTURE_HEADER, UCSAPI_PICTURE_HEADER_PTR;
```

Description:

A structure containing header information for photo data. The description of each value is as follows.

Format:

Contains the format information of the photo data. (Currently supports only the "JPG" format)

Specify the file extension value as a string.

Length:

It holds the size value of the photo data. The maximum data size that can be specified is 7K.

UCSAPI_PICTURE_DATA

Prototype:

```
typedef struct ucsapi_picture_data
{
    UCSAPI_PICTURE_HEADER          Header;
    UCSAPI_UINT8*                  Data;
} UCSAPI_PICTURE_DATA, UCSAPI_PICTURE_DATA_PTR;
```

Description:

A structure containing photo data information.

Header:

Contains the header information of the photo data.

Data:

A pointer to a buffer containing image data in the Format type of UCSAPI_PICTURE_HEADER. (Binary)

The resolution of the "JPG" data is 320 * 240.

UCSAPI_USER_COUNT

Prototype:

```
typedef struct ucsapi_user_count
{
    UCSAPI_UINT32 AdminNumber;
```

```
UCSAPI_UINT32 UserNumber;  
UCSAPI_USER_COUNT, *UCSAPI_USER_COUNT_PTR;
```

Description:

A structure that holds the number of users registered on the terminal.

Users are categorized as either administrators or regular users.

After calling the UCSAPI_GetUserCountFromTerminal function, UCSAPI_CALLBACK_EVENT_GET_USER_COUNT

You can obtain them at the event. The descriptions for each value are as follows.

AdminNumber:

It holds the number of registered administrators.

UserNumber :

Holds the number of registered general users.

UCSAPI_USER_INFO

Prototype:

```
typedef struct ucsapi_user_info  
{  
    UCSAPI_UINT32                UserID;  
    UCSAPI_DATA_PTR              UserName;  
    UCSAPI_DATA_PTR              UniqueID;  
    UCSAPI_USER_PROPERTY          Property;  
    UCSAPI_UINT8                 AuthType;  
    UCSAPI_ACCESS_FLAG            AccessFlag;  
    UCSAPI_ACCESS_AUTHORITY_PTR  AccessAuthority;  
    UCSAPI_ACU_PARTITION          Partition;  
    UCSAPI_USER_PROPERTY_EX       PropertyEx;  
    UCSAPI_UINT8                 Reserved[128];
```

```
UCSAPI_USER_INFO, UCSAPI_USER_INFO_PTR;
```

Description:

A structure containing user information. The description of each value is as follows.

UserID:

It holds user ID information. This value can only use numeric data of up to 8 digits.

UserName:

A pointer to a structure containing user name information. This value can be up to UCSAPI_DATA_SIZE_USER_NAME bytes. can be designated.

UniqueID:

A pointer to a structure containing the unique number (employee number) information. This value is used instead of UserID for user identification. It can be used. Up to UCSAPI_DATA_SIZE_UNIQUE_ID can be specified.

Property:

A structure containing user attribute information (authentication type and administrator status).

AuthType:

This value is no longer used due to the addition of the PropertyEx field. Always set it to 0.

Indicates authentication methods with values from 1 to 26 (1: Fingerprint, ..., 26: Card & Fingerprint & Facial & Password)

*For definitions of each value, refer to UCSAPI_Type.h

AccessFlag:

A structure containing bitmask information such as blacklists and face 1:N.

AccessAuthority:

A pointer to a structure containing access permission information.

Partition:

A structure containing the bitmask information for the accessible partitions of an ACU.

PropertyEx:

A structure containing user attribute (additional authentication method) information.

UCSAPI_USER_DATA

Prototype:

```
typedef struct ucsapi_user_data
{
    UCSAPI_USER_INFO           UserInfo;
    UCSAPI_AUTH_DATA_PTR       AuthData;
    UCSAPI_PICTURE_DATA_PTR   PictureData;
    UCSAPI_USER_DATA, UCSAPI_USER_DATA_PTR;
```

Description:

A structure containing user data. Used when calling UCSAPI_AddUserToTerminal. Each value corresponds to...

One explanation is as follows.

UserInfo:

A structure containing user information.

AuthData:

A pointer to a structure containing access permission data.

PictureData:

A pointer to a structure containing photo data.

UCSAPI_ACCESS_FLAG

Prototype:

```
typedef struct ucsapi_access_flag
{
    UCSAPI_UINT8    blacklist      :1;
    UCSAPI_UINT8    Face1toN     :1;
    UCSAPI_UINT8    reserved       :5;
    UCSAPI_UINT8    exceptpassback:1;

    UCSAPI_ACCESS_FLAG, UCSAPI_ACCESS_FLAG_PTR;
```

Description:

A structure containing additional user information.

blacklist:

The user's blacklist status holds a value. If blacklisted, it holds a value of 1; otherwise, it holds a value of 0.

has value..

Face1toN:

A bitmask holding the value indicating whether 1:N matching is possible during facial authentication. If 1:N matching is possible, it holds a value of 1.

In the case of n, it has a value of 0.

exceptpassback:

In environments using the anti-passback feature, users with this value set to 1 will not have the anti-passback feature applied.

does not.

UCSAPI_ERROR_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_ERROR_TYPE  
#define UCSAPI_ERROR_TYPE_NONE          0  
#define UCSAPI_ERROR_TYPE_ACCESS_LOG    1
```

Description:

Types of Error Types to be returned to the terminal for Callback Events received from the terminal defines.

3.1.4 Log-related types

It is declared in UCAPIL_Type.h and defines types related to authentication logs.

UCSAPI_GET_LOG_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_GET_LOG_TYPE;
```

Description:

To retrieve log data from the terminal, three types of log categories are defined.

The number of logs that can be stored internally varies depending on the terminal model, with a maximum of

When the log capacity is exceeded, some previous records are deleted to store new logs.

UCSAPI_GetAccessLogCountFromTerminal /

Used when calling UCSAPI_GetAccessLogFromTerminal.

```
#define UCSAPI_GET_LOG_TYPE_NEW          0  
#define UCSAPI_GET_LOG_TYPE_OLD          1  
#define UCSAPI_GET_LOG_TYPE_ALL          2  
#define UCSAPI_GET_LOG_TYPE_PERIOD      3
```

UCSAPI_LOG_IMAGE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_LOG_IMAGE;
```

Description:

To retrieve log images from the terminal, two types of log categories are defined.

Used when calling UCSAPI_GetAccessLogCountFromTerminalEx2.

```
#define UCSAPI_LOG_GET_IMAGE          0  
#define UCSAPI_LOG_NO_IMAGE          1
```

UCSAPI_ACCESS_LOG_DATA

Prototype:

```
typedef struct ucsapi_access_log_data  
{  
    UCSAPI_UINT32                UserID;  
    UCSAPI_DATE_TIME_INFO         DataTime;  
    UCSAPI_UINT8                 AuthMode;  
    UCSAPI_UINT8                 AuthType;  
    UCSAPI_UINT8                 DeviceID;  
    UCSAPI_UINT8                 ReaderID;  
    UCSAPI_BOOL                  IsAuthorized;  
    UCSAPI_DATA_PTR              RFID;  
    UCSAPI_PICTURE_DATA_PTR      PictureData;  
  
    UCSAPI_ACCESS_LOG_DATA, UCSAPI_ACCESS_LOG_DATA_PTR;
```

Description:

A structure containing authentication log data.

After calling UCSAPI_GetAccessLogFromTerminal, the UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG event

You can obtain it from the table. The descriptions for each value are as follows.

UserID:

It has a user ID value.

Date/Time :

A structure containing authentication timestamp information.

AuthMode:

Holds the authentication mode value. See the UCSAPI_AUTH_MODE definition.

AuthType:

Contains the authentication type value. Refer to the UCSAPI_AUTH_TYPE definition.

DeviceID

It holds the terminal type value. If this value is 0, it is the main terminal; if it is 1, it is the dummy reader.

ReaderID

It holds the ACU's ReaderID value. This value ranges from 0 to 7.

ReaderID:

Contains the authentication type value. Refer to the UCSAPI_AUTH_TYPE definition.

IsAuthorized:

It holds the authentication result value. It holds a value of 1 for success and a value of 0 for failure.

RFID:

Pointer to the structure containing the RFID data used during card authentication.

PictureData:

A pointer to a structure containing photo data captured during authentication. This value is available on devices capable of taking photos.

Only provided.

3.1.5 Callback-related types

UCAPI_Type.h declares and defines the types for callback events. The SDK receives from the terminal

A callback function is used to notify the application of incoming data.

Callback events are categorized into responses to application requests and requests from the terminal.

UCSAPI_CALLBACK_EVENT_HANDLER

Prototype:

```
typedef UCSAPI_RETURN (UCSAPI * UCSAPI_CALLBACK_EVENT_HANDLER) (
    UCSAPI_UINT32 TerminalID, UCSAPI_UINT32 EventType,
    UCSAPI_UINT32 wParam, UCSAPI_UINT32 lParam);
```

Description:

This is the definition of a callback function to receive events generated by the terminal.

EventType:

The second argument value distinguishes the event type.

wParam:

The third parameter value typically holds the UCSAPI_CALLBACK_PARAM_0_PTR value.

lParam:

The fourth argument typically holds the value UCSAPI_CALLBACK_PARAM_1_PTR.

UCSAPI_CALLBACK_PARAM_0

Prototype:

```
typedef struct ucsapi_callback_param_0
{
    UCSAPI_UINT32             ClientID;
    UCSAPI_UINT32             ErrorCode;
    UCSAPI_PROGRESS_INFO       Progress;
    UCSAPI_CALLBACK_PARAM_0, *UCSAPI_CALLBACK_PARAM_0_PTR;
```

Description:

The structure passed as the third argument to UCSAPI_CALLBACK_EVENT_HANDLER. For each value:

The explanation is as follows.

ClientID:

The client ID of the client who requested the work

ErrorCode:

It contains the value of an error that occurred during the executed task.

A value of 0 indicates success, while any other value indicates failure.

Progress:

A structure containing progress information for executed tasks, obtainable when issuing commands requiring progress indication.

UCSAPI_PROGRESS_INFO

Prototype:

```
typedef struct ucsapi_progress_info
{
    UCSAPI_UINT32          CurrentIndex;
    UCSAPI_UINT32          TotalNumber;
} UCSAPI_PROGRESS_INFO, *UCSAPI_PROGRESS_INFO_PTR;
```

Description:

A structure containing progress information for executed tasks. The UCS SDK sends multiple records to the application.

When notifying, include progress information in this structure along with the UCSAPI_CALLBACK_PARAM_0 structure.

Notify via the program.

UCSAPI_GetUserInfoListFromTerminal/UCSAPI_GetAccessLogFromTerminal/

It can be obtained after calling the UCSAPI_UpgradeFirmwareToTerminal function. The description of each value is as follows:

As follows.

CurrentIndex:

Index of the record currently being transmitted

TotalNumber:

Total number of records to be transmitted

UCSAPI_CALLBACK_PARAM_1

Prototype:

```
typedef struct ucsapi_callback_param_1
{
    UCSAPI_CALLBACK_DATA_TYPE          DataType;
    Union {
        UCSAPI_USER_INFO_PTR          UserInfo;
        UCSAPI_USER_DATA_PTR          UserData;
        UCSAPI_ACCESS_LOG_DATA_PTR    AccessLog;
        UCSAPI_FACE_INFO_PTR          FaceInfo;
    };
    Data;
} UCSAPI_CALLBACK_PARAM_1, *UCSAPI_CALLBACK_PARAM_1_PTR;
```

Description:

The structure passed as the fourth argument to UCSAPI_CALLBACK_EVENT_HANDLER. For each value:

The explanation is as follows.

DataType:

Specifies the data type of this structure. See UCSAPI_CALLBACK_DATA_TYPE.

Data:

A union structure specifying actual data. The values of UserInfo, UserData, and AccessLog are stored at a single address.

It is stored as a pointer for use.

UCSAPI_CALLBACK_DATA_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_CALLBACK_DATA_TYPE;
```

Description:

Specifies the data type of the UCSAPI_CALLBACK_PARAM_1 structure.

#define UCSAPI_CALLBACK_DATA_TYPE_USER_INFO	0
#define UCSAPI_CALLBACK_DATA_TYPE_USER_DATA	1
#define UCSAPI_CALLBACK_DATA_TYPE_ACCESS_LOG	2
#define UCSAPI_CALLBACK_DATA_TYPE_FACE_INFO	3

3.1.6 Access Control Setting Related Types

It is declared in UCAPIL_Type.h and defines types related to terminal access control.

UCSAPI_TIMEZONE

Prototype:

```
typedef struct ucsapi_timezone
{
    UCSAPI_TIME_HH_MM StartTime;
    UCSAPI_TIME_HH_MM EndTime;
} UCSAPI_TIMEZONE, * UCSAPI_TIMEZONE_PTR;
```

Description:

A structure containing time zone information.

StartTime / EndTime:

A structure containing time information from start to finish.

UCSAPI_ACCESS_TIMEZONE

Prototype:

```
typedef struct ucsapi_access_timezone
{
    UCSAPI_CHAR             Code[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_TIMEZONE         Zone[12];
    UCSAPI_UINT8             Reserved[4]
} UCSAPI_ACCESS_TIMEZONE, * UCSAPI_ACCESS_TIMEZONE_PTR;
```

Description:

A structure containing information about the time periods during which entry and exit are permitted throughout the day.

A maximum of 12 time zones can be assigned to a single time code. The description for each value is as follows:

It's the same.

Code:

A fixed-size string with the identifier code value for the time zone, UCSAPI_DATA_SIZE_CODE4(4).

Zone

An array of structures containing time zone information.

UCSAPI_ACCESS_TIMEZONE_DATA

Prototype:

```
typedef struct ucsapi_access_timezone_data
{
    UCSAPI_UINT32           TimezoneNum;
    UCSAPI_ACCESS_TIMEZONE  Timezone[UCSAPI_ACCESS_TIMEZONE_MAX];
} UCSAPI_ACCESS_TIMEZONE_DATA, * UCSAPI_ACCESS_TIMEZONE_DATA_PTR;
```

Description:

A structure containing data for accessible time slots. It can specify up to 128 time slot codes. Each

The descriptions of the values are as follows.

TimezoneNum:

Specify the number of codes for the total accessible time slots. The specified number of time slot information entries are contained in an array.
d.

Timezone:

An array of structures containing access time slot code information.

UCSAPI_ACCESS_HOLIDAY

Prototype:

```
typedef struct ucsapi_access_holiday
{
    UCSAPI_CHAR           Code[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_DATE_MM_DD     Date[32];
} UCSAPI_ACCESS_HOLIDAY, * UCSAPI_ACCESS_HOLIDAY_PTR;
```

Description:

A structure containing holiday information.

Up to 32 holidays can be designated in the holiday code. The description for each value is as follows:

As follows.

Code:

A fixed-size string with the identifier code value for holidays: UCSAPI_DATA_SIZE_CODE4(4).

Date

An array of structures containing holiday information.

UCSAPI_ACCESS_HOLIDAY_DATA

Prototype:

```
typedef struct ucsapi_access_holiday_data
{
    UCSAPI_UINT32          HolidayNum;
    UCSAPI_ACCESS_TIMEZONE   Holiday[UCSAPI_ACCESS_HOLIDAY_MAX];
} UCSAPI_ACCESS_HOLIDAY_DATA, * UCSAPI_ACCESS_HOLIDAY_DATA_PTR;
```

Description:

A structure containing holiday data.

Up to 64 holiday code data entries can be specified. The description for each value is as follows.

HolidayNum:

Specify the total number of holiday codes. The specified number of holiday entries are contained in an array.

Holiday:

An array of structures containing holiday code information.

UCSAPI_ACCESS_TIMEZONE_CODE

Prototype:

```
typedef struct ucsapi_access_timezone_code
{
    UCSAPI_CHAR           Sun[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           Mon[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           Tue[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           Wed[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           Thu[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           Fri[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           Sat[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           Hol[UCSAPI_DATA_SIZE_CODE4];
} UCSAPI_ACCESS_TIMEZONE_CODE, * UCSAPI_ACCESS_TIMEZONE_CODE_PTR;
```

Description:

A structure containing access time slot codes by day of the week. The descriptions for each value are as follows.

Sun / Mon / Tue / Wed / Thu / Fri / Sat:

It holds the code value for the accessible time slot by day of the week to be used during authentication.

Hol:

It holds the code value for the accessible time period to be applied during holidays at the time of authentication.

UCSAPI_ACCESS_TIME

Prototype:

```
typedef struct ucsapi_access_time
{
    UCSAPI_CHAR                     Code[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_ACCESS_TIMEZONE_CODE     Timezone;
    UCSAPI_CHAR                     Holiday[UCSAPI_DATA_SIZE_CODE4];
} UCSAPI_ACCESS_TIME, * UCSAPI_ACCESS_TIME_PTR;
```

Description:

A structure containing information about accessible hours. The description of each value is as follows.

Code:

A fixed-length string with a size of UCSAPI_DATA_SIZE_CODE4(4) representing the identifier code value for the accessible time.

Timezone:

It has a code value for the accessible time period by day of the week.

Holiday:

Contains the holiday code value to be used in the access time code.

The holiday code specified here corresponds to the time zone specified in the Hol field of the UCSAPI_ACCESS_TIMEZONE_CODE structure.

It is subject to the application of the principle.

UCSAPI_ACCESS_TIME_DATA

Prototype:

```
typedef struct ucsapi_access_time_data
{
    UCSAPI_UINT32                  AccessTimeNum;
    UCSAPI_ACCESS_TIME              AccessTime[UCSAPI_ACCESS_TIME_MAX];
} UCSAPI_ACCESS_TIME_DATA, * UCSAPI_ACCESS_TIME_DATA_PTR;
```

Description:

A structure containing data on accessible hours.

Up to 128 access time code entries can be specified. The description for each value is as follows:

It's the same.

AccessTimeNum:

Specify the number of total accessible time codes. The AccessTime information is entered into an array according to the number specified here.

There is.

AccessTime:

An array of structures containing the Code information for the verifiable time.

UCSAPI_ACCESS_GROUP

Prototype:

```
typedef struct ucsapi_access_group
{
    UCSAPI_CHAR           Code[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           AccessTime1[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           AccessTime2[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           AccessTime3[UCSAPI_DATA_SIZE_CODE4];
    UCSAPI_CHAR           AccessTime4[UCSAPI_DATA_SIZE_CODE4];
}
```

UCSAPI_ACCESS_GROUP, *UCSAPI_ACCESS_GROUP_PTR;

Description:

A structure containing access group code information. The descriptions for each value are as follows.

Up to four access time codes can be assigned to an access group code.

Code:

A fixed-length string with a size of UCSAPI_DATA_SIZE_CODE4(4) as the identifier code value for the access group.

AccessTime1 / AccessTime2 / AccessTime3 / AccessTime4:

Contains the access time code information to be used in the access group.

UCSAPI_ACCESS_GROUP_DATA

Prototype:

```
typedef struct ucsapi_access_group_data
{
    UCSAPI_UINT32          AccessGroupNum;
    UCSAPI_ACCESS_GROUP     AccessGroup[UCSAPI_ACCESS_GROUP_MAX];
}
```

UCSAPI_ACCESS_GROUP_DATA, * UCSAPI_ACCESS_GROUP_DATA_PTR;

Description:

A structure containing access group data. It can specify up to 128 access group codes.

The explanation for each value is as follows.

AccessGroupNum:

Specify the number of total access group codes. The AccessGroup information is stored in an array according to the number specified here.

d.

AccessGroup:

It is an array of structures containing access group code information.

UCSAPI_ACCESS_CONTROL_DATA_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_ACCESS_CONTROL_DATA_TYPE
```

#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_TIMEZONE	0
#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_HOLIDAY	1
#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_TIME	2
#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_GROUP	3

Description:

Specifies the data type of the data held by the UCSAPI_ACCESS_CONTROL_DATA structure.

UCSAPI_ACCESS_CONTROL_DATA

Prototype:

```
typedef struct ucsapi_access_control_data
{
    UCSAPI_ACCESS_CONTROL_DATA_TYPE          DataType;
    union {
        UCSAPI_ACCESS_TIMEZONE_DATA_PTR      Timezone;
        UCSAPI_ACCESS_HOLIDAY_DATA_PTR       Holiday;
        UCSAPI_ACCESS_TIME_DATA_PTR         AccessTime;
        UCSAPI_ACCESS_GROUP_DATA_PTR        AccessGroup;
    };
}
```

Data;

```
UCSAPI_ACCESS_CONTROL_DATA, * UCSAPI_ACCESS_CONTROL_DATA_PTR;
```

Description:

A structure containing access control information. Structure. Values for Timezone, Holiday, AccessTime, and AccessGroup are set.

It can be saved and used with my same address pointer.

Used in the UCSAPI_SetAccessControlDataToTerminal function.

The explanation for each value is as follows.

DataType:

Specifies the data type of this structure. See UCSAPI_ACCESS_CONTROL_DATA_TYPE.

3.1.7 Authentication-related types

It is declared in UCAPIL_Type.h and defines types related to user authentication.

UCSAPI_AUTH_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_AUTH_TYPE;
```

Description:

Defines the authentication type during user authentication.

#define UCSAPI_AUTH_TYPE_FINGER_1_TO_N	0
#define UCSAPI_AUTH_TYPE_FINGER_1_TO_1	1
#define UCSAPI_AUTH_TYPE_FINGER_CARD	2
#define UCSAPI_AUTH_TYPE_CARD	3
#define UCSAPI_AUTH_TYPE_PASSWORD	4
#define UCSAPI_AUTH_TYPE_FACE_1_TO_N	5
#define UCSAPI_AUTH_TYPE_FACE_1_TO_1	6

UCSAPI_AUTH_MODE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_AUTH_MODE;
```

Description:

Defines the authentication mode during user authentication. The authentication mode defines the purpose of the authentication. Generally, This can be applied when using the terminal for attendance purposes.

```
#define UCSAPI_AUTH_MODE_ATTENDANCE          1
#define UCSAPI_AUTH_MODE_LEAVE                 2
#define UCSAPI_AUTH_MODE_NORMAL               3
#define UCSAPI_AUTH_MODE_OUT                  4
#define UCSAPI_AUTH_MODE_RETURN              5
```

UCSAPI_INPUT_DATA_CARD

Prototype:

```
typedef struct ucsapi_input_data_card
{
    UCSAPI_UINT32           AuthMode;
    UCSAPI_DATA             RFID;
} UCSAPI_INPUT_DATA_CARD, *UCSAPI_INPUT_DATA_CARD_PTR;
```

Description:

A structure containing the information entered during Card authentication on the terminal. The description of each value is as follows.

AuthMode:

Holds the authentication mode value entered on the terminal. See UCSAPI_AUTH_MODE.

RFID:

A structure containing RFID information entered on the terminal.

UCSAPI_INPUT_DATA_PASSWORD

Prototype:

```
typedef struct ucsapi_input_data_password
{
    UCSAPI_UINT32           UserID;
    UCSAPI_UINT32           AuthMode;
    UCSAPI_DATA             Password;
} UCSAPI_INPUT_DATA_PASSWORD, *UCSAPI_INPUT_DATA_PASSWORD_PTR;
```

Description:

A structure containing the information entered during password authentication on the terminal. The description of each value is as follows:

It's the same.

UserID:

It possesses user ID information.

AuthMode:

Holds the authentication mode value. See UCSAPI_AUTH_MODE.

Password:

A structure containing password information.

UCSAPI_INPUT_DATA_FINGER_1_TO_1

Prototype:

```
typedef struct ucsapi_input_data_finger_1_to_n
{
    UCSAPI_UINT32             UserID;
    UCSAPI_UINT32             AuthMode;
    UCSAPI_UINT32             SecurityLevel;
    UCSAPI_DATA               Finger;
} UCSAPI_INPUT_DATA_FINGER_1_TO_1, *UCSAPI_INPUT_DATA_FINGER_1_TO_1_PTR;
```

Description:

A structure containing the information entered during 1:1 fingerprint authentication on the terminal. The description of each value is as follows:

It's the same.

UserID:

It possesses user ID information.

AuthMode:

Holds the authentication mode value. See UCSAPI_AUTH_MODE.

SecurityLevel:

It holds the security level value to be used during authentication.

The possible values are defined in the UCBioAPI_FIR_SECURITY_LEVEL definition of the UCBioBSP SDK.

Finger:

A structure containing fingerprint information.

UCSAPI_INPUT_DATA_FINGER_1_TO_N

Prototype:

```
typedef struct ucsapi_input_data_finger_1_to_n
{
    UCSAPI_UINT32             AuthMode;
    UCSAPI_UINT32             SecurityLevel;
    UCSAPI_UINT32             Input ID Length;
    UCSAPI_DATA               Finger;
} UCSAPI_INPUT_DATA_FINGER_1_TO_N, *UCSAPI_INPUT_DATA_FINGER_1_TO_N_PTR;
```

Description:

A structure containing the information entered during 1:N fingerprint authentication on the terminal. The description of each value is as follows:

It's the same.

UserID:

It possesses user ID information.

AuthMode:

Holds the authentication mode value. See UCSAPI_AUTH_MODE.

SecurityLevel:

It possesses a security level value to be used during authentication.

Input ID Length:

It holds the length value of the ID entered on the terminal. This value reduces the authentication scope during 1:N fingerprint authentication.

This can be used when seeking to improve authentication speed. The ID range for registered users is 0001 to 1000.

If the UserID value is 5 and the InputIDLength value is 2, the authentication ID range is 0500 to 1000.

It works.

Finger:

A structure containing fingerprint information entered on the terminal.

UCSAPI_INPUT_DATA_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_INPUT_DATA_TYPE;
```

#define UCSAPI_INPUT_DATA_TYPE_FINGER_1_TO_N	0
#define UCSAPI_INPUT_DATA_TYPE_FINGER_1_TO_1	1
#define UCSAPI_INPUT_DATA_TYPE_PASSWORD	2
#define UCSAPI_INPUT_DATA_TYPE_CARD	3
#define UCSAPI_INPUT_DATA_TYPE_FINGER_CARD	4

Description:

Specifies the data type of the data held by the UCSAPI_INPUT_DATA_TYPE structure.

UCSAPI_INPUT_DATA

Prototype:

```
typedef struct ucsapi_input_data
{
    UCSAPI_ANTIPASSBACK_LEVEL      Antipassback Level;
    UCSAPI_UINT8                    DeviceID;
    UCSAPI_INPUT_DATA_TYPE          DataType;
    Union {
        UCSAPI_INPUT_DATA_FINGER_1_TO_1_PTR Finger1To1;
        UCSAPI_INPUT_DATA_FINGER_1_TO_N_PTR Finger1ToN;
        UCSAPI_INPUT_DATA_CARD_PTR          Card;
        UCSAPI_INPUT_DATA_PASSWORD_PTR     Password;
    };
    Data;
    UCSAPI_UINT8                    ReaderID; // ACU ReaderID, The Value is 0~7
    UCSAPI_UINT8                    WiegandID; // ACU WiegandID, The Value is 1~4
    UCSAPI_ACU_DOOR_BIT_MASK        Door; // ACU door bit mask
}
```

```
UCSAPI_UINT8             Reserved[45];  
  
UCSAPI_INPUT_DATA, *UCSAPI_INPUT_DATA_PTR;  
  
Description:  
A structure containing information entered from the terminal during user authentication. When the terminal uses server authentication mode.  
This structure can store input information to request authentication from the application. Each value  
The explanation is as follows.
```

Antipassback Level:

It holds the anti-passback level value configured on the terminal. The application implements the anti-passback function.
The poem can refer to this value.

DataType:

Specifies the data type of the data held by this structure. See UCSAPI_INPUT_DATA_TYPE.

Data:

A union structure specifying actual data. Values of Finger1To1, Finger1ToN, Card, and Password are combined into a single
It is stored and can be used with the same address pointer.

Door:

A structure containing the bitmask value for the ACU door. This value indicates the door's status when the ACU requests service authentication.
This is necessary to check access permissions. The authentication server must grant access to doors where the corresponding bit is set to 1.
While checking access permissions, when sending the authentication result (SendAuthResultToTerminal), the terminal must have access permissions.
Set to 1 for doors with access permission, and reset to 0 for doors without access permission.

UCSAPI_INPUT_ID_TYPE

Prototype:

```
typedef UCSAPI_UINT32          UCSAPI_INPUT_ID_TYPE;  
  
#define UCSAPI_INPUT_ID_TYPE_USER_ID           0  
#define UCSAPI_INPUT_ID_TYPE_UNIQUE_ID         1  
#define UCSAPI_INPUT_ID_TYPE_RFID              2
```

Description:

Specify the type of ID entered on the terminal. The ID type entered during 1:1 authentication is determined by the terminal option settings.

It can be set. The default value uses the UCSAPI_INPUT_ID_TYPE_USER_ID type. User ID value

The maximum usable range is up to 8 digits, so if this is exceeded,

Using the UCSAPI_INPUT_ID_TYPE_UNIQUE_ID type allows for a maximum of 20 characters.

UCSAPI_INPUT_ID_DATA

Prototype:

```
typedef struct ucsapi_input_id_data
{
    UCSAPI_INPUT_ID_TYPE          DataType;
    Union {
        UCSAPI_UINT32*            UserID;
        UCSAPI_DATA_PTR           UniqueID;
        UCSAPI_DATA_PTR           RFID;
    };
    Data;
} UCSAPI_INPUT_ID_DATA, *UCSAPI_INPUT_ID_DATA_PTR;
```

Description:

A structure containing the ID information entered from the terminal during user authentication on the server. Settings for each value.

The names are as follows.

DataType:

Specifies the data type of this structure. See UCSAPI_INPUT_ID_TYPE.

Data:

A union structure specifying actual data. UserID, UniqueID, and RFID values are stored under a single address pointer.

It is available for saving and use.

UCSAPI_AUTH_INFO

Prototype:

```
typedef struct ucsapi_auth_info
{
    UCSAPI_UINT32                UserID;
    UCSAPI_BOOL                  IsAccessibility;
```

```
UCSAPI_USER_PROPERTY          Property;
UCSAPI_UINT32                  ErrorCode;
UCSAPI_AUTH_INFO, *UCSAPI_AUTH_INFO_PTR;
```

Description:

A structure containing the user's authentication information. Used in the UCSAPI_SendAuthInfoToTerminal function.

The explanation for each value is as follows.

UserID:

It holds the user's ID value.

IsAccessibility:

Holds a value indicating whether the user can be authenticated. Returns a value of 1 for success and 0 for failure.

has.

Property:

A structure containing user attribute information (authentication type, administrator status).

ErrorCode:

If the user lacks authentication privileges, it returns the corresponding error code value. Refer to the error code table.

UCSAPI_AUTH_NO_NOTIFICATION

Prototype:

```
typedef struct ucsapi_auth_notify
{
    UCSAPI_UINT32          UserID;
    UCSAPI_BOOL             IsAuthorized;
    UCSAPI_BOOL             IsVisitor;
    UCSAPI_DATE_TIME_INFO   AuthorizedTime;
    UCSAPI_UINT32            ErrorCode;
} UCSAPI_AUTH_NOTIFY, *UCSAPI_AUTH_NOTIFY_PTR;
```

Description:

A structure containing the user's authentication result information.

Used in the UCSAPI_SendAuthResultToTerminal function.

UserID:

It holds the ID value of a user who has been authenticated or attempted to authenticate.

IsAuthorized:

It holds the authentication result value. It holds a value of 1 upon success and a value of 0 upon failure.

IsVisitor:

The value indicating whether a visitor is an authenticated user is held. It holds a value of 1 upon success and 0 upon failure.

It has value.

Authorized Time:

A structure containing authentication timestamp information.

ErrorCode:

Upon authentication failure, it returns an error code value. Refer to the error code table.

3.1.8 Terminal Option Setting Related Types

It is declared in UCAPIL_Type.h and defines types related to terminal optional settings.

UCSAPI_TERMINAL_TIMEZONE

Prototype:

```
typedef struct ucsapi_terminal_timezone
{
    UCSAPI_UINT8        IsUsed;
    UCSAPI_UINT8        StartHour;
    UCSAPI_UINT8        StartMin;
    UCSAPI_UINT8        EndHour;
    UCSAPI_UINT8        EndMin;
} UCSAPI_TERMINAL_TIMEZONE, * UCSAPI_TERMINAL_TIMEZONE_PTR;
```

Description:

A structure containing time information used in the device's lock/unlock schedule. Description of each value:

The following is as follows.

IsUsed:

It validates the values held by the UCSAPI_TERMINAL_TIMEZONE structure.

StartHour / StartMin:

It has start time information.

EndHour / EndMin:

It has end time information.

UCSAPI_TERMINAL_DAY_SCHEDULE

Prototype:

```
typedef struct ucsapi_terminal_day_schedule
{
    UCSAPI_TERMINAL_TIMEZONE           Lock1;
    UCSAPI_TERMINAL_TIMEZONE           Lock2;
    UCSAPI_TERMINAL_TIMEZONE           Lock3;
    UCSAPI_TERMINAL_TIMEZONE           Open1;
    UCSAPI_TERMINAL_TIMEZONE           Open2;
    UCSAPI_TERMINAL_TIMEZONE           Open3;
} UCSAPI_TERMINAL_DAY_SCHEDULE, * UCSAPI_TERMINAL_DAY_SCHEDULE_PTR;
```

Description:

A structure containing the lock/unlock schedule information for terminals by day of the week.

The schedule per day allows you to set up to three lock/unlock time slots per day. Each value

The explanation of the fields is as follows.

Lock1 / Lock2 / Lock3:

It has a time slot value for locking the device during the day.

Open1 / Open2 / Open3:

It holds a time slot value for opening the terminal during the day.

UCSAPI_HOLIDAY_TYPE

Prototype:

```
typedef UCSAPI_UINT8 UCSAPI_HOLIDAY_TYPE;
```

```
#define UCSAPI_HOLIDAY_TYPE_1      1  
#define UCSAPI_HOLIDAY_TYPE_2      2  
#define UCSAPI_HOLIDAY_TYPE_3      3
```

Description:

Specify the holiday type to be used in the device lock/unlock schedule. Up to three holiday types can be specified.

The following is an explanation of each value.

UCSAPI_TERMINAL_HOLIDAY_INFO

Prototype:

```
typedef struct ucsapi_holiday_info  
{  
    UCSAPI_UINT8        Month;  
    UCSAPI_UINT8        Day;  
    UCSAPI_UINT8        HolidayType;  
} UCSAPI_TERMINAL_HOLIDAY_INFO, * UCSAPI_TERMINAL_HOLIDAY_INFO_PTR
```

Description:

A structure containing holiday information.

Month/Day:

Contains information about holiday dates.

Holiday type:

Contains the holiday type value. See UCSAPI_HOLIDAY_TYPE.

UCSAPI_TERMINAL_SCHEDULE

Prototype:

```
typedef struct ucsapi_terminal_schedule
{
    UCSAPI_TERMINAL_DAY_SCHEDULE Sun;
    UCSAPI_TERMINAL_DAY_SCHEDULE Mon;
    UCSAPI_TERMINAL_DAY_SCHEDULE Tue;
    UCSAPI_TERMINAL_DAY_SCHEDULE Wed;
    UCSAPI_TERMINAL_DAY_SCHEDULE Thu;
    UCSAPI_TERMINAL_DAY_SCHEDULE Fri;
    UCSAPI_TERMINAL_DAY_SCHEDULE Sat;
    UCSAPI_TERMINAL_DAY_SCHEDULE Holiday1;
    UCSAPI_TERMINAL_DAY_SCHEDULE Holiday2;
    UCSAPI_TERMINAL_DAY_SCHEDULE Holiday3;
    UCSAPI_TERMINAL_HOLIDAY_INFO Holidays[100];
}
```

UCSAPI_TERMINAL_SCHEDULE, * UCSAPI_TERMINAL_SCHEDULE_PTR;

Description:

A structure containing the device's lock/unlock schedule information by day of the week.

Holidays can be set up to a maximum of 100, and their types can be designated as Holiday1, Holiday2, or Holiday3.

The following is an explanation of each value.

Sun / Mon / Tue / Wed / Thu / Fri / Sat:

A structure containing lock/unlock schedule information by day of the week.

Holiday1 / Holiday2 / Holiday3:

A structure containing lock/unlock schedule information by holiday type.

Holidays:

A structure containing holiday information. Each holiday is scheduled on one of Holiday1, Holiday2, or Holiday3.

Receive the dragon.

UCSAPI_SECURITY_LEVEL

Prototype:

```
typedef struct ucsapi_security_level
{
    UCSAPI_UINT8     Verify          :4; /* 1:1 default level = 4 */
    UCSAPI_UINT8     Identify        :4; /* 1:N default level = 5 */
} UCSAPI_SECURITY_LEVEL, * UCSAPI_SECURITY_LEVEL_PTR;
```

Description:

A structure containing security level information used during fingerprint authentication. This value can take the following values.

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

Verify:

1:1 authentication level value. The default value is 4.

Identify

1:N authentication level value. The default value is 5.

UCSAPI_ANTIPASSBACK_LEVEL

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_ANTIPASSBACK_LEVEL;
```

#define UCSAPI_ANTIPASSBACK_LEVEL_NOT_USE	0
#define UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTATION_ALLOW	1
#define UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTATION_PROHIBIT	2

Description:

It holds the level value of the anti-passback feature configured on the terminal. The terminal uses the anti-passback function.

To enable UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTON_ALLOW or

It must be set to UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTON_PROHIBIT.

[UCSAPI_ANTIPASSBACK_LEVEL_NOT_USE:](#)

Anti-passback not in use.

[UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTON_ALLOW:](#)

Access permitted while disconnected from the server.

[UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTON_PROHIBIT:](#)

Access is prohibited while the connection to the server is disconnected.

UCSAPI_NETWORK_INFO

Prototype:

```
typedef struct ucsapi_network_info
{
    UCSAPI_UINT8          NetworkType;
    UCSAPI_UINT8          IP[4];
    UCSAPI_UINT8          Subnet[4];
    UCSAPI_UINT8          Gateway[4];
    UCSAPI_NETWORK_INFO;
```

Description:

A structure containing the terminal's network configuration information.

[NetworkType:](#)

It holds the type value of the IP address. If the value is 0, it supports a static IP; if it is 1, it supports a dynamic IP.

[IP:](#)

An array of buffers containing the terminal's IP address values.

[Subnet:](#)

An array of buffers containing Subnet Mask values.

Gateway:

An array of buffers containing gateway address values.

UCSAPI_SERVER_INFO

Prototype:

```
typedef struct ucsapi_server_info
{
    UCSAPI_UINT8                IP[4];
    UCSAPI_UINT16               Port;
    UCSAPI_UINT8                Reserved[2];
} UCSAPI_SERVER_INFO;
```

Description:

A structure containing network information for the terminal to connect to the server. Each value is described as follows:

It is like sound.

IP:

This is a buffer array containing the address values of the server IP.

Port:

It holds the socket port value for server connection.

UCSAPI_TERMINAL_OPTION_FLAG

Prototype:

```
typedef struct ucsapi_terminal_option_flag
{
    UCSAPI_UINT32        SecurityLevel      :1;
    UCSAPI_UINT32        Input ID Length    :1;
    UCSAPI_UINT32        AutoEnterKey      :1;
    UCSAPI_UINT32        Sound             :1;
    UCSAPI_UINT32        Authentication     :1;
    UCSAPI_UINT32        Application       :1;
    UCSAPI_UINT32        Antipassback      :1;
}
```

```

UCSAPI_UINT32          Network           :1;
UCSAPI_UINT32          Server            :1;
UCSAPI_UINT32          Input ID Type    :1;
UCSAPI_UINT32          AccessLevel      :1;
UCSAPI_UINT32          PrintText        :1;
UCSAPI_UINT32          Schedule          :1;

UCSAPI_TERMINAL_OPTION_FLAG *UCSAPI_TERMINAL_OPTION_FLAG_PTR;

```

Description:

Holds the reference flag value for each item in the UCSAPI_TERMINAL_OPTION structure. The corresponding item's

You can only reference the value of an item when the flag value is True.

For descriptions of each item, refer to the documentation for the UCSAPI_TERMINAL_OPTION structure.

UCSAPI_TERMINAL_OPTION

Prototype:

```

typedef struct ucsapi_terminal_option
{
    UCSAPI_TERMINAL_OPTION_FLAG      Flags;
    UCSAPI_SECURITY_LEVEL           SecurityLevel;
    UCSAPI_UINT8                    InputIDLength;
    UCSAPI_UINT8                    AutoEnterKey;
    UCSAPI_UINT8                    Sound;
    UCSAPI_UINT8                    Authentication;
    UCSAPI_UINT8                    Application;
    UCSAPI_UINT8                    Antipassback;
    UCSAPI_NETWORK_INFO             Network;
    UCSAPI_SERVER_INFO              Server;
    UCSAPI_UINT8                    InputIDType;
    UCSAPI_UINT8                    AccessLevel;
    UCSAPI_UINT8                    PrintText[32];
    UCSAPI_TERMINAL_SCHEDULE        Schedule;

UCSAPI_TERMINAL_OPTION, *UCSAPI_TERMINAL_OPTION_PTR;

```

Description:

A structure that holds the terminal's option settings.

Used in the UCSAPI_SetOptionToTerminal / UCSAPI_GetOptionFromTerminal functions. Each value

The explanation of the fields is as follows.

Flags:

It holds the reference flag value for each item in the structure.

To set terminal option items using the UCSAPI_SetOptionToTerminal function, you must

Set the flag value of the item to True and specify the item value.

SecurityLevel:

Specify the security level to use during authentication.

The possible values are defined in the UCBioAPI_FIR_SECURITY_LEVEL definition of the UCBioBSP SDK.

InputIDLength:

The terminal holds the length value of the input ID. When using UserID, a maximum of 8 can be specified, and when using UniqueID,

You can set the maximum level to 20.

AutoEnterKey:

Holds the value indicating whether the terminal's auto-enter key is enabled. This function detects Key input for the duration specified by InputIDLength

This feature automatically inputs the Enter key when you press it.

Sound:

Holds the sound volume value of the terminal. The volume value can be set from 0 to 20. The terminal's sound

To set it to mute, specify 0.

Authentication:

It holds the authentication method value of the terminal.

The possible values are 1.5 pages. Refer to the "Terminal Authentication Method" section in the glossary.

Application:

The terminal program has a mode value. Terminals are categorized by function into access control, attendance, and meal services, etc.

It can be used. For possible values, refer to "Terminal Program Mode" in Section 1.6 Terminology.

Antipassback:

It holds the anti-passback level value of the terminal.

The possible values are defined in UCSAPI_ANTIPASSBACK_LEVEL.

Network:

A structure containing the terminal's network information.

Server:

A structure containing network information for the terminal to connect to the server.

Input ID Type:

It holds the type value of the ID entered during authentication on the terminal. The possible values are as follows.

0 – UserID

1 – UniqueID

AccessLevel:

It has an access level value. This function restricts the authentication types that can be entered on the terminal, allowing only specified types.

Enables authentication. Possible values are as follows. The default value is 0.

0 – No restrictions.

1 – Fingerprint and password authentication only.

PrintText:

This is a buffer array containing the string to be printed on the water printer connected to the terminal.

This value can be used when a water printer is connected to the terminal.

Schedule: A structure containing lock/unlock schedule information.

UCSAPI_ACU_OPTION

Prototype:

```
#define MAX_CP040_READER          12
#define MAX_CP040_PARTITION         4
#define MAX_CP040_ZONE              8
#define MAX_CP040_PGM                8
#define MAX_CP040_DOOR               4
#define MAX_CP040_INPUT               4
```

```

typedef struct ucsapi_acu_option
{
    ACU_NET_SETTING           netSettings;
    ACU_READER_OPTION          reader[MAX_CP040_READER];
    ACU_PARTITION_INFO          part[MAX_CP040_PARTITION];
    ACU_ZONE_CONFIG             zone[MAX_CP040_ZONE];
    ACU_PROGRAM_OPTION          pgm[MAX_CP040_PGM];
    ACU_DOOR_OPTION              door[MAX_CP040_DOOR];
    ACU_INPUT_OPTION              inputs[MAX_CP040_INPUT];
    ACU_SYSTEM_OPTION             sysOpt;
    ACU_UDP_SETTING                 udpset;
    BYTE                         resv[11];

UCSAPI_ACU_OPTION, *UCSAPI_ACU_OPTION_PTR;

```

Description:

A structure containing the option settings for ACU.

Used in the UCSAPI_SetOptionToACU / UCSAPI_GetOptionFromACU functions.

For detailed items, please refer to the sample source.

UCSAPI_ACU_LOCK_SCHEDULE

Prototype:

```
#define MAX_ACU_LOCK 4
```

```

typedef struct ucsapi_acu_lockschedule
{
    UCSAPI_UINT8                LockIndex; // 0 - 3
    UCSAPI_TERMINAL_SCHEDULE      Schedule;

UCSAPI_ACU_LOCKSCHEDULE, *UCSAPI_ACU_LOCKSCHEDULE_PTR;

```

Description:

A structure containing the schedule settings for the designated lock of the ACU.

Used in the UCSAPI_SetLockScheduleToACU / UCSAPI_GetLockScheduleFromACU functions.

For UCSAPI_TERMINAL_SCHEDULE, refer to TerminalOption.

UCSAPI_PERIPHERAL_DEVICE

Prototype:

```
typedef struct ucsapi_peripheral_device
{
    UCSAPI_SINT32          DeviceID;
    UCSAPI_SINT32          ControlCommand;
} UCSAPI_PERIPHERAL_DEVICE, *UCSAPI_PERIPHERAL_DEVICE_PTR;
```

Description:

A structure containing commands to be transmitted to peripheral devices. Used when calling UCSAPI_ControlPeripheralDevice.

It is possible. The explanation for each value is as follows.

DeviceID:

Peripheral Device ID.

ControlCommand :

Peripheral device control command.

UCSAPI_SET_EMERGENCY_INFO

Prototype:

```
typedef struct      ucsapi_set_emergency_info
{
    UCSAPI_EMERGENCY_TYPE          SignalType;
    UCSAPI_EMERGENCY_VALUE          SignalValue;
    UCSAPI_EMERGENCY_DOOR_CONTROL   Door;
    UCSAPI_EMERGENCY_ALARM_CONTROL  Alarm;
    UCSAPI_UINT8                   Reserved[28];
} UCSAPI_SET_EMERGENCY_INFO, *UCSAPI_SET_EMERGENCY_INFO_PTR;
```

Description:

A structure containing emergency-related settings. Used when calling UCSAPI_SetEmergencyToTerminal. Each

The descriptions of the values are as follows.

SignalType :

FIRE – 1

PANIC - 2

SignalValue:

START - 1

STOP - 0

Door:

NONE - 0

OPEN - 1

Alarm:

NONE - 0

PLAY - 1.

UCSAPI_VOIP_INFO

Prototype:

```
typedef struct ucsapi_voip_info{  
    char ProxyAddr[256];  
    char LogonID[32];  
    char LogonPwd[32];  
    UCSAPI_UINT8 Reserved[2048];  
    UCSAPI_VOIP_INFO *UCSAPI_VOIP_INFO_PTR;
```

Description:

A structure containing VoIP terminal settings.

Used when calling UCSAPI_GetVoipInfoFromTerminal and UCSAPI_SetVoipInfoToTerminal. Each

The explanation of the values is as follows.

Proxy Address:

VoIP terminal address

LogonID:

Account ID

Logon Password:

Account Password

3.1.9 Monitoring-related types

It is declared in UCAPI_Type.h and defines monitoring-related types.

UCSAPI_TERMINAL_STATUS

Prototype:

```
typedef struct ucsapi_terminal_status
{
    UCSAPI_UINT32 Terminal;
    UCSAPI_UINT32 Door;
    UCSAPI_UINT32 Cover;
    UCSAPI_UINT32 Lock;
    UCSAPI_UINT32 Open;
    UCSAPI_UINT32 Reserved1;
    UCSAPI_UINT32 Reserved2;
    UCSAPI_UINT32 Reserved3;
} UCSAPI_TERMINAL_STATUS, *UCSAPI_TERMINAL_STATUS_PTR;
```

Description:

A structure containing the terminal's status values. The descriptions for each value are as follows.

Terminal:

Holds the lock status value of the terminal. Possible values are as follows.

0 – Unlock

1 – Lock

2 – Shutdown (Global locking)

Door:

Holds the lock status value of the terminal. This value is only supported on locks that support monitoring functionality.

It is possible. The possible values are as follows.

0 – Close

1 – Open

2 – Do Not Use

3 – Forced Open

4 – Not Closed

Cover:

Holds the cover status value of the terminal. Possible values are as follows.

0 – Close

1 – Open

Lock:

Displays the status value of the door lock. Possible values are as follows.

0 – Normal

1 – Error

Open:

Indicates the open state of the entrance door. Possible values are as follows.

0 – Normally Open

1 – Continuous Open

UCSAPI_ACU_STATUS_INFO

Prototype:

```
#define MAX_ACU_PARTITION 4
#define MAX_ACU_ZONE          8
#define MAX_ACU_LOCK           4
#define MAX_ACU_READER          8
```

```
typedef struct acu_reader_ver
{
    BYTE hw;
    BYTE major;
    BYTE minor;
    BYTE custom1;
    BYTE custom2;
    BYTE order;
    BYTE reserved[2];
} ACU_READER_VER;
```

```
typedef struct acu_reader
{
    BYTE id;
```

```

BYTE reader_type;
ACU_READER_VER ver; // 8
ACU_READER;

typedef struct acu_status
{
    BYTE partition[MAX_ACU_PARTITION];
    BYTE zone[MAX_ACU_ZONE];
    BYTE lock[MAX_ACU_LOCK];
    BYTE reader[MAX_ACU_READER];
    ACU_READER reader_ver[MAX_ACU_READER];
    BYTE Reserved[8];
} ACU_STATUS;

typedef struct ucsapi_acu_status_info
{
    UCSAPI_UINT8      Notice;
    ACU_STATUS        Status;
} UCSAPI_ACU_STATUS_INFO, *UCSAPI_ACU_STATUS_INFO_PTR;

```

Description:

A structure containing the status values of the ACU terminal. The descriptions for each value are as follows.

For descriptions of each item, please refer to the ACU manual.

UCSAPI_TERMINAL_CONTROL

Prototype:

```

typedef struct ucsapi_terminal_control
{
    UCSAPI_UINT8      lockStatus;
    UCSAPI_UINT8      lockType;
} UCSAPI_TERMINAL_CONTROL, *UCSAPI_TERMINAL_CONTROL_PTR;

```

Description:

A structure containing values for terminal control. The descriptions for each value are as follows.

lockStatus:

Holds the terminal's lock status value. Possible values are as follows.

0 – Unlock

1 – Lock

lockType:

Specifies the type of device lock. Possible values are as follows.

0 – Normal

1 – Global (Shutdown)

3.2 API References

This document describes the definitions of various APIs used in the UCS SDK, along with their function usage and parameters.

Among API arguments, those that are commonly applied

ClientID is the ID of the client that requested the task and is used in Client/Server model development.

TerminalID refers to the ID of the terminal requesting the task.

3.2.1 General API

This is a description of the APIs for starting and stopping the UCS SDK.

UCSAPI_ServerStart

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_ServerStart(
    IN UCSAPI_UINT32 MaxTerminal,
    IN UCSAPI_UINT32 Port,
    IN UCSAPI_INT32 Reserved,
    IN UCSAPI_CALLBACK_EVENT_HANDLER CallBackEventFunction);
```

Description:

Initializes the UCSAPI module and executes server functions.

Parameters:

MaxTerminal:

Defines the maximum number of connected terminals. The SDK adjusts internal memory usage based on the entered maximum terminal count.

Pre-allocating memory can improve speed. If the number of connected devices exceeds the maximum, memory is automatically

Increase usage to secure connections.

Port:

Communication port for terminal connection. The server waits for the terminal's connection on the specified port. The default value is

9870. If you change this value, you must also change the terminal's port value.

Callback Event Function:

Pointer to the callback function for event notifications in the application

Returns:

UCSAPIERR_NONE
UCSAPIERR_FUNCTION_FAILED

Callback:

UCSAPI_CALLBACK_EVENT_CONNECTED

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0 CallBack0

lParam:

UCSAPI_UINT8 TerminalIP[4]:

An array of buffers with terminal IP addresses

`UCSAPI_ServerStop`

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_ServerStop();
```

Description:

Disconnect all connected terminals and terminate server functions.

Parameters:

N/A.

Returns:

`UCSAPIERR_NONE`

Callback:

N/A

Callback Parameters:

N/A.

UCSAPI_SetTerminalTimezone

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetTerminalTimezone(
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_CHAR_PTR pTimezoneName);
```

Description:

Set the time zone for the connected device.

The terminal generally uses the time zone of the connected server. However, the terminal's time zone and

If the server's time zone is incorrect, the terminal's time zone must be set to the local time zone.

Parameters:

TerminalID:

Device ID

pTimezoneName:

Specify the name of the time zone to change.

The time zone name can be found under the following registry key.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\TimeZones

Returns:

UCSAPIERR_NONE

Callback:

N/A

Callback Parameters:

N/A.

※ GMT stands for

When referring to standard time zones, GMT is generally used as a reference.

GMT stands for Greenwich Mean Time, with Greenwich, England serving as the starting point for standard time.

is used.

For example, Korea is GMT+9, which means it is 9 hours ahead of Greenwich Mean Time.

The place with the earliest time is Wellington, New Zealand. Since it's three hours ahead of Korea, the time in Wellington is...

It is 10 o'clock in the morning.

New York is GMT-5, so here it's 5 p.m. the previous day.

※ Time zone Name

The time zone name can be referenced from the following registry entry.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\TimeZones

	이름	종류	데이터
	(기본값)	REG_SZ	(값 설정 안됨)
	Display	REG_SZ	(GMT-09:00) 알래스카
	Dst	REG_SZ	알래스카 일왕 혈약 시간제
	Index	REG_DWORD	0x00000003 (3)
	MapID	REG_SZ	30.31
	Std	REG_SZ	알래스카 표준시
	TZI	REG_BINARY	1c 02 00 00 00 00 c4 ff ff 00 00 0b 00

UCSAPI_SetError

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetError(  
    IN UCSAPI_UINT32 TerminalID,  
    IN UCSAPI_ERROR_TYPE ErrorType);
```

Description:

Configuration to return an error code to the terminal for a Callback Event received from the terminal does.

Parameters:

TerminalID:

Device ID

ErrorType:

It holds the Error Type value for returning to the terminal.

For example, after calling UCSAPI_GetAccessLogFromTerminal, store the received log data in the database.

If the operation fails, set the ErrorType to UCSAPI_ERROR_TYPE_ACCESS_LOG within the Callback Event function.

When UCSAPI_SetError is called, the terminal sets a new log entry for the most recently transmitted log.

It will be retained and sent again upon the next New Log request.

Returns:

UCSAPIERR_NONE

Callback:

N/A

Callback Parameters:

N/A.

UCSAPI_SetWiegandFormatToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetWiegandFormatToTerminal(
    IN UCSAPI_UINT32 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_WIEGAND_DATA wgdType,
    IN UCSAPI_CHAR_PTR FilePath);
```

Description:

Configure the terminal's Wiegand In/Out Format. The Wiegand Format File is provided with the SDK.

It can be generated using a Wiegand Tool.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

wgdType:

It has a Type value to distinguish between Wiegand In and Wiegand Out.

#define UCSAPI_WIEGAND_DATA_TYPE_OUT	1
#define UCSAPI_WIEGAND_DATA_TYPE_IN	2

FilePath:

Full path to the Wiegand Format Data File

Returns:

UCSAPIERR_NONE

Callback:

UCSAPI_CALLBACK_EVENT_SET_WIEGAND_FORMAT

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

N/A

UCSAPI_SetFingerImageSend

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetFingerImageSend (
    IN UCSAPI_BOOL IsSend);
```

Description:

Enable or disable the fingerprint image reception function on the terminal.
(Available only when fingerprint authentication is enabled on the server)

Parameters:

IsSend:

Fingerprint Image Reception Function Enabled/Disabled

Returns:

UCSAPIERR_NONE

Callback:

N/A

Callback Parameters:

N/A.

3.2.2 Terminal User Management API

This document describes the API for managing terminal users.

UCSAPI_AddUserToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_AddUserToTerminal(  
    IN UCSAPI_INT32 ClientID,  
    IN UCSAPI_INT32 TerminalID,  
    IN UCSAPI_BOOL IsOverwrite,  
    IN UCBioAPI_USER_DATA_PTR* pUserData);
```

Description:

Transfers user information to the designated terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

IsOverwrite:

If the user is already registered, specify whether to overwrite. The default value is 1.

pUserData:

A pointer to a structure containing user data.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

UCSAPIERR_USER_NAME_SIZE
UCSAPIERR_UNIQUE_ID_SIZE
UCSAPIERR_INVALID_SECURITY_LEVEL
UCSAPIERR_INVALID_PARAMETER
UCS API ERROR CODE SIZE
UCSAPIERR_PASSWORD_SIZE
UCSAPIERR_MAX_CARD_NUMBER
UCSAPIERR_MAX_FINGER_NUMBER
UCSAPIERR_PICTURE_SIZE

Callback:

UCSAPI_CALLBACK_EVENT_ADD_USER

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_UINT32 UserID;

※ Precautions

When using the UCSAPI_AddUserToTerminal function to send multiple user data entries to the terminal, the function returns a value.

After calling UCSAPI_AddUserToTerminal, check for the UCSAPI_CALLBACK_EVENT_ADD_USER event.

After confirming that it has been processed normally, the next user data must be transmitted.

UCSAPI_DeleteUserFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_DeleteUserFromTerminal(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID,
    IN UCSAPI_INT32 UserID);
```

Description:

Deletes user data from the specified device.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID

UserID:

The ID of the user you wish to delete

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_DELETE_USER

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_UINT32 UserID;

※ Precautions

When deleting multiple users from a terminal using the UCSAPI_DeleteUserFromTerminal function, you must

After calling UCSAPI_DeleteUserFromTerminal, check for the UCSAPI_CALLBACK_EVENT_DELETE_USER event.

After confirming that it was processed normally, the next user must be deleted.

UCSAPI_DeleteAllUsersFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_DeleteAllUsersFromTerminal(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID);
```

Description:

Delete all user data from the specified device.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_DELETE_ALL_USERS

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

N/A

UCSAPI_GetUserCountFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetUserCountFromTerminal(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID);
```

Description:

Retrieves the number of registered users from the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the work..

TerminalID:

Device ID.

Returns:

```
UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL
```

Callback:

UCSAPI_CALLBACK_EVENT_GET_USER_COUNT

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_UINT32 nUserCount

It contains the number of users.

UCSAPI_GetUserInfoListFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetUserInfoListFromTerminal(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID);
```

Description:

Retrieves the list of all registered user information from the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Returns:

```
UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL
```

Callback:

UCSAPI_CALLBACK_EVENT_GET_USER_INFO_LIST

Callback Parameters:

wParam:

```
UCSAPI_CALLBACK_PARAM_0_PTR           pCallback0;
```

lParam:

```
UCSAPI_CALLBACK_PARAM_1_PTR           pCallback1
pCallback1.DataType = UCSAPI_CALLBACK_DATA_TYPE_USER_INFO
```

UCSAPI_GetUserDataFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetUserDataFromTerminal(  
    IN UCSAPI_INT32 ClientID,  
    IN UCSAPI_UINT32 TerminalID  
    IN UCSAPI_UINT32 UserID);
```

Description:

Retrieves user data from the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

UserID:

User ID.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_USER_DATA

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

```
UCSAPI_CALLBACK_PARAM_1_PTR pCallback1;  
pCallback1.DataType = UCSAPI_CALLBACK_DATA_TYPE_USER_DATA
```

UCSAPI_RegisterFaceFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_RegisterFaceFromTerminal (  
    IN UCSAPI_UINT16 ClientID,  
    IN UCSAPI_UINT32 TerminalID,  
    IN UCSAPI_UINT8 opt);
```

Description:

When this command is issued, it will prompt for facial input from the designated terminal.

Until the stop command is issued, standard registration captures 5 faces and quick registration captures 3 faces.

An event occurs each time a capture is made.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

opt:

Command options: 0: Start registration, 1: Stop registration.

Returns:

```
UCSAPIERR_NONE  
UCSAPIERR_NOT_SERVER_ACTIVE  
UCSAPIERR_INVALID_POINTER
```

Callback:

UCSAPI_CALLBACK_EVENT_REGISTER_FACE

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;
o When the TotalNumber and CurrentIndex values of Progress are 0, it indicates that the input has been canceled.

lParam:

Upon normal face input, receives the address of the face data structure (UCSAPI_FACE_INFO_PTR)

UCSAPI_RegisterWalkThroughFaceFromTerminal

Prototype:

```
UCSAPI RETURN UCSAPI UCSAPI_RegisterWalkThroughFaceFromTerminal (
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_UINT8 opt);
```

Description:

When this command is issued, facial input will be received from the designated terminal.

It captures one face until the stop command is issued, and an event occurs upon completion.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

opt:

Command options: 0: Start registration, 1: Stop registration.

Returns:

WSEAPIERR_NONE
WSEAPIERR_ERROR
UCSAPIERR_INVALID_TERMINAL

Callback 1:

UCSAPI_CALLBACK_EVENT_GET_WALKTHROUGH_JPG

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

ClientID and ErrorCode can be verified.

lParam:

UCSAPI_CALLBACK_PARAM_1_PTR pCallback1;

pCallback1.DataType = UCSAPI_CALLBACK_DATA_TYPE_WALKTHROUGH_JPG

pCallback1.Data.WalkThrough = UserID, Type(JPG or Template), Length(buffer size),

Data (image buffer)

Callback 2:

UCSAPI_CALLBACK_EVENT_GET_WALKTHROUGH_TEMPLATE

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

ClientID and ErrorCode can be verified.

lParam:

UCSAPI_CALLBACK_PARAM_1_PTR pCallback1;

pCallback1.DataType = UCSAPI_CALLBACK_DATA_TYPE_WALKTHROUGH_TEMPLATE

pCallback1.Data.WalkThrough = UserID, Type(JPG or Template), Length(buffer size),

Data (Template buffer)

UCSAPI_RegisterIrisFromTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_RegisterIrisFromTerminal (

IN UCSAPI_UINT16 ClientID,

IN UCSAPI_UINT32 TerminalID,

IN UCSAPI_UINT8 opt);

Description:

When this command is issued, the iris will be captured from the designated terminal.
After touching the start button on the terminal, the iris scan begins within 10 seconds.
After touching the start button on the terminal, a 10-second timeout is applied.
Even if you do not touch the start button on the terminal, it has a 60-second timeout.
The function will stop when it times out or when a stop command is issued from the server.
An event occurs when scanning is complete or stopped.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

opt:

Command options: 0: Start registration, 1: Stop registration.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER

Callback:

UCSAPI_CALLBACK_EVENT_REGISTER_IRIS

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

- When the TotalNumber and CurrentIndex values of Progress are 0, it indicates that the input has been canceled.

lParam:

Upon normal iris input, receives the iris data structure address (UCSAPI_IRIS_INFO_PTR)

UCSAPI_EnrollFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_EnrollFromTerminal (
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    OUT UCBioAPI_EXPORT_DATA_PTR pFingerData);
```

Description:

When this command is issued, a fingerprint will be requested from the designated terminal.

It will scan the fingerprint twice until the stop command is issued.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

pFingerData:

A pointer to the structure that will hold the scanned Template data.

UCBioBSP SDK's UCBioAPI_EXPORT_DATA Structure Reference..

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_USER_CANCEL
UCSAPIERR_FUNCTION_FAIL
UCSAPIERR_INVALID_TERMINAL

UCSAPI_FreeExportData

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_FreeExportData (
    IN UCBioAPI_EXPORT_DATA_PTR pFingerData);
```

Description:

Free the memory allocated for the UCBioAPI_EXPORT_DATA structure.

Parameters:

pFingerData:

Pointer to the UCBioAPI_EXPORT_DATA structure to be released.

If this value is NULL, do nothing.

Returns:

UCSAPIERR_NONE

3.2.3 Log-related API

This document describes the API for obtaining log data stored on the terminal.

UCSAPI_GetAccessLogCountFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLogCountFromTerminal(  
    IN UCSAPI_INT32 ClientID,  
    IN UCSAPI_INT32 TerminalID,  
    IN UCSAPI_GET_LOG_TYPE LogType);
```

Description:

Retrieves the number of authentication logs from the specified terminal.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

LogType:

Specifies the log type to retrieve. Possible values are listed in UCSAPI_GET_LOG_TYPE.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_PARAMETER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG_COUNT

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_UINT32 nLogCount;

It contains the number of logs.

UCSAPI_GetAccessLogCountFromTerminalEx

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLogCountFromTerminal(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID,
    IN UCSAPI_GET_LOG_TYPE LogType,
    IN UCSAPI_DATE_PERIOD_PTR Period);
```

Description:

GetAccessLogCountFromTerminal is an extended API. Retrieve log information by period.

Parameters for setting the duration have been added.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

LogType:

Specifies the log type to retrieve. Possible values are listed in UCSAPI_GET_LOG_TYPE.

Period:

A pointer to a structure containing date information for the log data to be retrieved. Retrieve log information by period.

This value is used when logging. It is used when LogType is set to UCSAPI_LOG_TYPE_PERIOD.

Returns:
UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_PARAMETER
UCSAPIERR_INVALID_TERMINAL

Callback:
UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG_COUNT

Callback Parameters:

wParam:
UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:
UCSAPI_UINT32 nLogCount;

It contains the number of logs.

UCSAPI_GetAccessLogFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLogFromTerminal(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID,
    IN UCSAPI_GET_LOG_TYPE LogType);
```

Description:

Retrieves authentication log data from the designated terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

LogType:

Specifies the log type to retrieve. Possible values are listed in UCSAPI_GET_LOG_TYPE.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_PARAMETER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR	pCallback0;						
Log	Record	Various	7	in the case of	Record's	Number	as much as

The UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG event occurs. The total number of logs and the current log

The Index information refers to the pCallback0->Progress value.

Param:

```
UCSAPI_CALLBACK_PARAM_1_PTR           pCallback1;  
pCallback1->DataTyp = UCSAPI_CALLBACK_DATA_TYPE_ACCESS_LOG;
```

UCSAPI_GetAccessLogFromTerminalEx

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLogFromTerminalEx(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID,
    IN UCSAPI_GET_LOG_TYPE LogType,
    IN UCSAPI_DATE_PERIOD_PTR Period);
```

Description:

This is an extended API for the GetAccessLogFromTerminal function. Retrieve log information by period.

Parameters for setting the duration have been added.

Parameters:

ClientID:

The client ID of the client who requested the work.

TerminalID:

Device ID.

LogType:

Specifies the log type to retrieve. Possible values are listed in UCSAPI_GET_LOG_TYPE.

Period:

A pointer to a structure containing date information for the log data to be retrieved. Retrieve log information by period.

This value is used when logging. It is used when LogType is set to UCSAPI_LOG_TYPE_PERIOD.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_PARAMETER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG

Callback Parameters:

wParam:

The UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG event occurs. The total number of logs and the current log

The Index information refers to the pCallback0->Progress value.

IParam:

UCSAPI_CALLBACK_PARAM_1_PTR pCallback1;
pCallback1->DataType = UCSAPI_CALLBACK_DATA_TYPE_ACCESS_LOG;

UCSAPI_GetAccessLogFromTerminalEx2

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLogFromTerminalEx2(
    IN UCSAPI_INT32 ClientID,
    IN UCSAPI_INT32 TerminalID,
    IN UCSAPI_GET_LOG_TYPE LogType,
    IN UCSAPI_LOG_IMAGE LogImage,
    IN UCSAPI_DATE_PERIOD_PTR Period);
```

Description:

This is an extended API for the UCSAPI_GetAccessLogFromTerminalEx function. Retrieve log images.

A parameter has been added for.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

LogType:

Specifies the log type to retrieve. Possible values are listed in UCSAPI_GET_LOG_TYPE.

LogImage:

Specifies whether to use log images. Possible values are defined in UCSAPI_LOG_IMAGE.

Period:

A pointer to a structure containing date information for the log data to be retrieved. Retrieve log information by period.

This value is used when logging. It is used when LogType is set to UCSAPI_LOG_TYPE_PERIOD.

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_PARAMETER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallBack0;

Log	Record	Various	기호	in the case of	Record's	Number	as much as

The UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG event occurs. The total number of logs and the current log

The Index information refers to the pCallback0->Progress value.

IParam:

UCSAPI_CALLBACK_PARAM_1_PTR pCallback1;
pCallback1->DataType = UCSAPI_CALLBACK_DATA_TYPE_ACCESS_LOG;

3.2.4 Authentication-related API

This describes the API for performing authentication on the server.

When the authentication method of the terminal is N/S or NO mode, the terminal requests authentication from the server.

UCSAPI_SendAuthenticationInformationToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SendAuthInfoToTerminal(  
    IN UCSAPI_INT32 TerminalID,  
    IN UCSAPI_AUTH_INFO_PTR pUserAuthInfo);
```

Description:

The terminal 1:1 CertificationCity User Certificationinformation obtaining for Applicationby program

Notifies the UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO event.

At this point, the application places the user's authentication information into the UCSAPI_AUTH_INFO structure and sends it to the terminal immediately.

The poem must be sent.

The UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO event always occurs before the following events.

occurs.

```
UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1,  
UCSAPI_CALLBACK_EVENT_VERIFY_CARD,  
CSAPI_CALLBACK_EVENT_VERIFY_PASSWORD
```

Parameters:

TerminalID:

Device ID.

pUserAuthInfo:

A pointer to a structure containing the user's authentication information.

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:
UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO

Callback Parameters:

wParam:
UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:
UCSAPI_INPUT_ID_DATA_PTR plInputID;

A structure containing the ID information entered from the terminal.

UCSAPI_SendAntipassbackResultToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SendAntipassbackResultToTerminal(  
    IN UCSAPI_INT32 TerminalID,  
    IN UCSAPI_INT32 UserID,  
    IN UCSAPI_BOOL bResult);
```

Description:

When the terminal has the Antipassback option enabled and performs authentication, the user's Notify the application of this event to obtain the current Antipassback status. At this time, the application The program must immediately send the user's antipassback status to the terminal in bResult. This A vent occurs only when the terminal performs authentication on the terminal.

Parameters:

TerminalID:

Device ID.

UserID:

User ID.

bResult:

Access permission status based on antipassback condition. If access is permitted, it holds a value of 1.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_ANTIPASSBACK

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_UINT32 UserID;

UCSAPI_SendAuthResultToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SendAuthResultToTerminal(  
    IN UCSAPI_INT32 TerminalID,  
    IN UCSAPI_AUTH_NOTIFY_PTR pResult);
```

Description:

The terminal notifies the application of the following events for user authentication.

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N (1:N Fingerprint Verification Request)

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1 (1:1 Fingerprint Verification Request)

UCSAPI_CALLBACK_EVENT_VERIFY_CARD (Card Authentication Request)

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD (Password Authentication Request)

At this point, the application places the user authentication result into a UCSAPI_AUTH_NOTIFY structure and immediately sends it to the terminal.

The poem must be sent.

Parameters:

TerminalID:

Device ID.

pResult:

A pointer to a structure containing the authentication result.

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1

UCSAPI_CALLBACK_EVENT_VERIFY_CARD

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_INPUT_DATA_PTR plInputData;

A pointer to a structure containing sample data input from the terminal.

3.2.5 Terminal Management API

This document describes the API for device management.

UCSAPI_GetTerminalCount

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetTerminalCount(  
    IN UCSAPI_UINT32* pTerminalCount);
```

Description:

Retrieve the number of terminals connected to the server.

Parameters:

pTerminalCount:

A pointer to the value that will hold the number of terminals.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER

Callback:

N/A

Callback Parameters:

N/A

`UCSAPI_GetFirmwareVersionFromTerminal`

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetFirmwareVersionFromTerminal(
    IN UCSAPI_UINT32 ClientID,
    IN UCSAPI_UINT32 TerminalID);
```

Description:

Retrieves the firmware version of the specified device.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:

`UCSAPI_CALLBACK_EVENT_FW_VERSION`

Callback Parameters:

wParam:

`UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;`

lParam:

`UCSAPI_DATA_PTR pVersion;`

Pointer to a structure containing version information

UCSAPI_UpgradeFirmwareToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI_UCSAPI_UpgradeFirmwareToTerminal(
    IN UCSAPI_UINT32 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_CHAR_PTR pFilePath);
```

Parameters:

Upgrade the firmware of the specified device. Upgrade progress information is notified via the following events.

UCSAPI_CALLBACK_EVENT_FW_UPGRADING / UCSAPI_CALLBACK_EVENT_FW_UPGRADED

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

pFilePath:

Pointer to the value containing the path of the firmware file.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_FW_UPGRADING
UCSAPI_CALLBACK_EVENT_FW_UPGRADED

Callback Parameters:

wParam:

For upgrade progress information, refer to the pCallback0->Progress structure.

IParam:

N/A

UCSAPI_SendUserFileToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI_UCSAPI_SendUserFileToTerminal (
```

IN UCSAPI_UINT32 ClientID,
IN UCSAPI_UINT32 TerminalID,
IN UCSAPI_UINT32 FileType,
IN UCSAPI_CHAR_PTR FilePath);

Parameters:

Download the specified user file to the designated device.

Parameters:

ClientID: 1.6 See Glossary

TerminalID: 1.6 See Glossary

FileType

- 1: Text file (.csv),
 - 2: Wallpaper (.jpg),
 - 3: Success Audio File (.wav),
 - 4: False Negative File (.wav),
 - 5: Video file (.mp4)

FilePath:

This is the absolute path value for the user file

Returns:

IJCSAP19

UCCARRIER NOT

UCCAPIERR_INVALID_POINTER

000A1E98_INVALID_POINTER

Copyright 2009 UNIONCOMMUNITY Co., LTD.

Callback:

UCSAPI_CALLBACK_EVENT_USERFILE_UPGRADING : Download in progress
UCSAPI_CALLBACK_EVENT_USERFILE_UPGRADED: After download completion

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0 pCallback0;

For download progress information, refer to the `pCallback0->Progress` structure.

IParam:

N/A

UCSAPI_GetOptionFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetOptionFromTerminal(
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID);
```

Description:

Retrieves the option setting value from the specified terminal.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback Parameters:

UCSAPI_CALLBACK_EVENT_GET_TERMINAL_OPTION

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0 pCallback0;

lParam:

UCSAPI_TERMINAL_OPTION_PTR pOption;

UCSAPI_SetOptionToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetOptionToTerminal(
    IN UCSAPI_UINT32 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_TERMINAL_OPTION_PTR pOption);
```

Description:

Set the option value for the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

pOption:

Pointer to the UCSAPI_TERMINAL_OPTION structure.

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_SET_TERMINAL_OPTION

wParam

UCSAPI_CALLBACK_PARAM_0_PTR pCallBack0;

lParam:

N/A

UCSAPI_OpenDoorToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_OpenDoorToTerminal(
    IN UCSAPI_UINT32 ClientID,
    IN UCSAPI_UINT32 TerminalID);
```

Description:

Temporarily unlocks the specified device.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_OPEN_DOOR

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

N/A

UCSAPI_OpenDoorToTerminalEx

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_OpenDoorToTerminalEx(  
    IN UCSAPI_UINT32 ClientID,  
    IN UCSAPI_UINT32 TerminalID,  
    IN UCSAPI_UINT32 OpenTime);
```

Description:

This is an extended API for the UCSAPI_OpenDoorToTerminal function. It relates to the door open hold time.

A parameter has been added for.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

OpenTime:

Door open duration.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_OPEN_DOOR

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

N/A

UCSAPI_ControlPeripheralDevice

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_ControlPeripheralDevice(  
    IN UCSAPI_UINT32 ClientID,  
    IN UCSAPI_UINT32 TerminalID,  
    IN UCSAPI_PERIPHERAL_DEVICE* PeripheralDevice);
```

Description:

Control the peripheral devices of the designated terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Peripheral Device:

Specify the peripheral device ID and command. Possible values are listed in UCSAPI_PERIPHERAL_DEVICE.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_CONTROL_PERIPHERAL_DEVICE

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_PERIPHERAL_DEVICE_PTR pCallback1;

UCSAPI_SetDoorStatusToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetDoorStatusToTerminal(
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_UINT32 Status);
```

Description:

Controls the lock mechanism of the designated terminal according to the Status value.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Status:

Lock Status (0: Temporary Open, 1: Lock Released, 2: Lock Corrected)

Returns:

```
UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL
```

Callback:

UCSAPI_CALLBACK_EVENT_OPEN_DOOR

Callback Parameters:

wParam:

```
UCSAPI_CALLBACK_PARAM_0_PTR           pCallback0;
```

lParam:

N/A

UCSAPI_SendTerminalControl

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SendTerminalControl (
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_UINT32 lockStatus,
    IN UCSAPI_UINT32 lockType);
```

Description:

Controls the specified terminal to match the designated value. Global Lock shuts down the terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

lockStatus:

Lock Status (0: Unlock, 1: Lock)

lockType:

Lock Type (0:Normal, 1:Global)

Returns:

UCSAPIERR_NONE

Callback:

UCSAPI_CALLBACK_EVENT_TERMINAL_CONTROL

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

UCSAPI_TERMINAL_CONTROL pCtrl;

`UCSAPI_SetAccessControlDataToTerminal`

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetAccessControlDataToTerminal(
    IN UCSAPI_UINT32 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_ACCESS_CONTROL_DATA_PTR pAccessControlData);
```

Description:

Transmits access control information to the designated terminal. Time periods, access times, holidays, and access group information are each separately

The access control information must be transmitted over the road. It is used for authentication at the terminal. The terminal uses the stored access control information.

If no information is available, no separate access control is applied.

Parameters:

ClientID, TerminalID: See parent document

pAccessControlData:

A pointer to a structure containing access control information.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:

`UCSAPI_CALLBACK_EVENT_SET_ACCESS_CONTROL_DATA`

Callback Parameters:

wParam:

`UCSAPI_CALLBACK_PARAM_0_PTR` pCallback0;

lParam:

N/A

UCSAPI_GetTerminalInfo

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetTerminalInfo(
    IN UCSAPI_UINT32 TerminalID,
    OUT UCSAPI_TERMINAL_INFO_PTR pInfo);
```

Description:

Acquire additional information for the specified terminal. Additional information includes IP Address, MAC Address, etc. Various information is available; for details, please refer to the structure.

Parameters:

ClientID, TerminalID: See parent document

UCSAPI_TERMINAL_INFO_PTR:

A pointer to a structure containing terminal information.

```
typedef struct ucsapi_terminal_info
{
    UCSAPI_UINT32 TerminalID;
    UCSAPI_UINT8     TerminalIP[4];
    UCSAPI_UINT8     TerminalStatus;
    UCSAPI_UINT8     DoorStatus;
    UCSAPI_UINT8     CoverStatus;
    UCSAPI_UINT8     LockStatus;
    UCSAPI_UINT8     ExtSignal[4]; // Contact signals 1, 2, 3, 4 (Reserved)
    UCSAPI_VERSION Firmware;
    UCSAPI_VERSION Protocol;
    UCSAPI_VERSION CardReader;
    UCSAPI_UINT16 ModelNo;
    UCSAPI_UINT8     TerminalType;
    UCSAPI_UINT8     MacAddress[6];
}
```

UCSAPI_TERMINAL_INFO, *UCSAPI_TERMINAL_INFO_PTR;

Returns:

UCSAPIERR_NONE

UCSAPIERR_INVALID_TERMINAL

Callback: None

UCSAPI_SendPrivateMessageToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SendPrivateMessageToTerminal(
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_UINT32 Reserved,
    IN UCSAPI_PRIVATE_MESSAGE_PTR Message)
```

Description:

Sends a specific phrase to be displayed on the terminal's screen.

The Message structure contains the time (in seconds) and the message to be displayed.

In the usual case, when authentication succeeds, a message is sent to notify the user.

Parameters:

ClientID, TerminalID: See parent document

Message: Structure pointer of the output phrase

```
typedef struct ucsapi_private_message
{
    UCSAPI_UINT16           messageSize;
    UCSAPI_UINT16           displayTime;
    UCSAPI_CHAR             messageData[UCSAPI_MESSAGE];
    UCSAPI_PRIVATE_MESSAGE, *UCSAPI_PRIVATE_MESSAGE_PTR;
```

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_PRIVATE_MESSAGE

Callback Parameters:

wParam:
UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

N/A

UCSAPI_SendPublicMessageToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SendPublicMessageToTerminal(  
    IN UCSAPI_UINT16 ClientID,  
    IN UCSAPI_UINT32 TerminalID,  
    IN UCSAPI_BOOL Show,  
    IN UCSAPI_PUBLIC_MESSAGE_PTR Message)
```

Description:

Command the terminal to display a notification on its screen at the designated time during the specified period.

The Message structure contains the period, time, and phrase to be output.

Parameters:

ClientID, TerminalID: See parent document

Message: Structure pointer of the output phrase

```
typedef struct ucsapi_public_message  
{  
    UCSAPI_UINT16             Reserved1;  
    UCSAPI_UINT8              Reserved2[2];  
    UCSAPI_DATE_YYYY_MM_DD    StartDate;  
    UCSAPI_DATE_YYYY_MM_DD    EndDate;  
    UCSAPI_TIME_HH_MM          StartTime;  
    UCSAPI_TIME_HH_MM         EndTime;  
    UCSAPI_CHAR                Message[UCSAPI_MESSAGE];  
    UCSAPI_PUBLIC_MESSAGE,*UCSAPI_PUBLIC_MESSAGE_PTR;
```

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_PUBLIC_MESSAGE

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

N/A

UCSAPI_SendSirenToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SendSirenToTerminal(  
    IN UCSAPI_UINT16 ClientID,  
    IN UCSAPI_UINT32 TerminalID,  
    IN UCSAPI_UINT8 opt,  
    IN UCSAPI_UINT8 cntSiren,  
    IN UCSAPI_SIREN_CONFIG_PTR pSiren)
```

Description:

Retrieves siren data configured from the terminal or transmits configured siren data to the terminal.

When obtaining siren data (opt=0), subsequent parameter values are ignored.

Parameters:

ClientID, TerminalID: See parent document

opt:

Command Issue Option (0: Get siren data, 1: Write siren data)

cntSiren:

The number of UCSAPI_SIREN_CONFIG values included in pSiren (maximum value is 100)

pSiren: Siren Data Structure

```

#define MAX_SIREN_CONFIG          100
struct WeekDay_Flag
{
    BYTE Sunday           :1;
    BYTE Monday           :1;
    BYTE Tuesday          :1;
    BYTE Wednesday         :1;
    BYTE Thursday          :1;
    BYTE Friday            :1;
    BYTE Saturday          :1;
    BYTE OffHoliday        :1;
};

typedef struct siren_config
{
    BYTE                Hour;
    BYTE                Minute;
    Weekday Flag        wf;
    BYTE                Duration;      // Seconds
    BYTE                Reserved[4];
UCSAPI_SIREN_CONFIG, *UCSAPI_SIREN_CONFIG_PTR;

```

At Hour:Minute, the siren sounds for Duration (in seconds).

Recurring tasks can be scheduled by day of the week and can exclude public holidays.

Callback:

`UCSAPI_CALLBACK_EVENT_GET_SIREN`: When the opt value is 0

`UCSAPI_CALLBACK_EVENT_SET_SIREN`: When the opt value is 1

Callback Parameters:

wParam:

`UCSAPI_CALLBACK_PARAM_0_PTR` pCallback0;

When the opt value is 0, pCallback0->Progress.TotalNumber represents the number of subsequent siren data points.

lParam:

When the opt value is 0, it has the `UCSAPI_SIREN_CONFIG_PTR` value.

UCSAPI_SetSmartCardLayoutToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetSmartCardLayoutToTerminal (
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_SMARTCARD_LAYOUT_PTR pCardLayout)
```

Description:

Transmit the card layout data to the terminal to set the card layout that will be read by the terminal.

Parameters:

ClientID, TerminalID: See parent document
pCardLayout: Card Layout Data Structure

```
#define UCSAPI_SMARTCARD_SECTOR_MAX          128
typedef UCSAPI_UINT32 UCSAPI_SMARTCARD_KEYTYPE;
#define UCSAPI_SMARTCARD_KEYTYPE_A            0x60
#define UCSAPI_SMARTCARD_KEYTYPE_B            0x61

typedef struct ucsapi_smartcard_sector_layout
{
    UCSAPI_UINT8           SectorNumber;           // Sector Number (0..127)
    UCSAPI_SMARTCARD_KEYTYPE KeyType;             // Key A = 0x60, Key B = 0x61
    UCSAPI_UINT8           KeyData[6];             // Key Value
    UCSAPI_UINT8           BlockNumber;           // Block Number (0..3)
    UCSAPI_UINT8           Start;                 // Start Location in Block
    UCSAPI_UINT8           Length;                // Data Length
    UCSAPI_UINT8           AID[2];                // AID of MAD Card. If AID not used, set 0xff
    // (This field is used for MAD Card. If the byte value is 0xff, this byte is not used.)
    UCSAPI_UINT8           Reserved[3];
}

UCSAPI_SMARTCARD_SECTOR_LAYOUT, *UCSAPI_SMARTCARD_SECTOR_LAYOUT_PTR;

typedef UCSAPI_UINT32 UCSAPI_SMARTCARD_TYPE;
#define UCSAPI_SMARTCARD_TYPE_DATA            0
#define UCSAPI_SMARTCARD_TYPE_FINGER         1
```

```

typedef UCSAPI_UINT32 UCSAPI_SMARTCARD_READTYPE;
#define UCSAPI_SMARTCARD_READTYPE_SERIAL          0
#define UCSAPI_SMARTCARD_READTYPE_DATA           1
#define UCSAPI_SMARTCARD_READTYPE_MAD            2

typedef UCSAPI_UINT8 UCSAPI_SMARTCARD_SERIALFORMAT;
#define UCSAPI_SMARTCARD_SERIALFORMAT_DEFAULT     0
#define UCSAPI_SMARTCARD_SERIALFORMAT_HEXA       1
#define UCSAPI_SMARTCARD_SERIALFORMAT_DECIMAL    2
#define UCSAPI_SMARTCARD_SERIALFORMAT_35DECIMAL  3

typedef struct ucsapi_smartcard_layout
{
    UCSAPI_SMARTCARD_TYPE             CardType;
    UCSAPI_SMARTCARD_READ_TYPE        ReadType;
    UCSAPI_SMARTCARD_SERIAL_FORMAT   SerialFormat;

    // If ReadType is UCSAPI_SMARTCARD_READTYPE_SERIAL, this value is valid data.
    UCSAPI_UINT8                      SectorNumber; // 0..127
    UCSAPI_UINT8                      Reserved[6];
    UCSAPI_SMARTCARD_SECTOR_LAYOUT*   Layouts[UCSAPI_SMARTCARD_SECTOR_MAX];

    // Maximum card size is 8k
    UCSAPI_SMARTCARD_LAYOUT*          pLayout;
};

Callback:
UCSAPI_CALLBACK_EVENT_SET_SMARTCARD_LAYOUT

Callback Parameters:
wParam:
UCSAPI_CALLBACK_PARAM_0_PTR         pCallback0;

lParam:
N/A

```

UCSAPI_GetFingerprintMinutiaeFromTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_GetFpMinutiaeFromTerminal (
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_UINT8 minType,
    IN UCSAPI_UINT8 minCount)
```

Description:

Request the terminal to capture a fingerprint, extract its feature points, and transmit the feature point information to the server.

Some devices may not yet support this feature, so please verify before use.

Parameters:

ClientID, TerminalID: See parent document

minType: Feature point information (Currently only supports 0:UNION Type)

minCount: Number of fingerprint inputs (currently supports only two inputs)

Callback:

```
UCSAPI_CALLBACK_EVENT_FP_MINUTIAE
```

Callback Parameters:

wParam:
UCSAPI_CALLBACK_PARAM_0_PTR pCallback0;

lParam:

minutiae information

```
typedef struct ucsapi_fp_minutiae
{
    BYTE minType; // Default 0
    BYTE minCount; // Def 2
    BYTE matching; // Matching Result
    long lenData; // Real Data Length (minData Size == Def:800)
    BYTE* minData; // Fingerprint Minutiae Data
} UCSAPI_FP_MINUTIAE, *UCSAPI_FP_MINUTIAE_PTR;
```

UCSAPI_TerminalOptionDialog

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_TerminalOptionDialog (
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID);
```

Description:

Displays a dialog providing terminal settings-related functions for the specified terminal.
(Windows platform only)

Parameters:

[ClientID: 1.6 See term explanation](#)

[TerminalID: 1.6 See Terminology Explanation](#)

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback Parameters:

UCSAPI_CALLBACK_EVENT_GET_TERMINAL_OPTION

wParam

UCSAPI_CALLBACK_PARAM_0_PTR pCallBack0;

lParam

UCSAPI_TERMINAL_OPTION_PTR pOption

UCSAPI_SetEmergencyToTerminal

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_SetEmergencyToTerminal (
    IN UCSAPI_UINT16 ClientID,
    IN UCSAPI_UINT32 TerminalID,
    IN UCSAPI_SET_EMERGENCY_INFO_PTR Emergency);
```

Description:

Configure the emergency-related settings for the designated terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Emergency:

Configure the functions to be used in case of an emergency.

The possible values are listed in UCSAPI_SET_EMERGENCY_INFO.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback Parameters:

UCSAPI_CALLBACK_EVENT_SET_EMERGENCY

wParam

UCSAPI_CALLBACK_PARAM_0_PTR pCallBack0;

lParam:

N/A

UCSAPI_RebootToTerminal

Includes note[w2]: Added 20241227

Prototype:

```
UCSAPI_RETURN UCSAPI UCSAPI_RebootToTerminal(  
    IN UCSAPI_UINT16 ClientID,  
    IN UCSAPI_UINT32 TerminalID);
```

Description:

Force a reboot of the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Returns:

```
UCSAPIERR_NONE  
UCSAPIERR_NOT_SERVER_ACTIVE  
UCSAPIERR_INVALID_TERMINAL
```

Callback Parameters:

UCSAPI_CALLBACK_EVENT_TERMINAL_REBOOT

wParam

UCSAPI_CALLBACK_PARAM_0_PTR pCallBack0;

lParam:

N/A

3.3 Callback Event References

UCAPI_Type.h declares and defines the types for callback events. The SDK receives from the terminal

A callback function is used to notify the application of incoming data.

Callback events are categorized into responses to application requests and requests from the terminal.

3.3.1 Events for Requests from the Terminal

An event issued at the terminal's request, typically to notify the terminal of its status or to request the server's response regarding result processing.
does.

UCSAPI_CALLBACK_EVENT_CONNECTED

Description:

The SDK module notifies the application of this event when the device connects to the server.

UCSAPI_CALLBACK_EVENT_DISCONNECTED

Description:

The SDK module notifies the application when the device's connection is terminated.

UCSAPI_CALLBACK_EVENT_TERMINAL_STATUS

Description:

The SDK module periodically or upon state changes gathers status information about the terminal and devices attached to the terminal's surroundings

In the event of an error, immediately notify the server of the status.

UCSAPI_CALLBACK_EVENT_ACU_STATUS

Description:

The SDK module periodically or upon status changes provides status information for the ACU (MCP-040, etc.) terminal and connected devices.

Notify the server immediately of any changes to its status.

Callback Parameters:

Param:

UCSAPI_ACU_STATUS_INFO_PTR

UCSAPI_CALLBACK_EVENT_GET_TERMINAL_TIME

Description:

The SDK module enables the API to set the current time when the device requests it.

It triggers a vent. If not configured in the API, the SDK reports the current system time.

If there is no event handling or the handling result value is UCSAPI_CALLBACK_RESULT_DEFAULT,

The SDK currently allows the system time to be set on the device.

Callback Parameters:

IPParam:

UCSAPI_DATE_TIME_INFO_PTR: Address value of the structure containing the current time

3.3.2 Events of Response for Server Command

In response to commands sent from the terminal, the server typically issues commands first.

UCSAPI_CALLBACK_EVENT_REALTIME_ACCESS_LOG

Description:

The UCS SDK module operates in S/N mode and performs user authentication on the device when this occurs.

Notify the event to the application.

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG

Description:

The SDK module responds to authentication log requests stored on the application's terminal with this event.

Notifies the application.

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG_COUNT

Description:

The SDK module responds to the application's request for the number of authentication logs stored on the terminal.

Notify the event to the application.

UCSAPI_CALLBACK_EVENT_ADD_USER

Description:

The SDK module sends this event to the application in response to the application's user addition request.

We hereby notify you.

UCSAPI_CALLBACK_EVENT_DELETE_USER

Description:

The SDK module sends this event to the application in response to the application's user deletion request.

We hereby notify you.

UCSAPI_CALLBACK_EVENT_DELETE_ALL_USERS

Description:

The SDK module responds to this event in the application in response to all user deletion requests.

Notify via RAM.

UCSAPI_CALLBACK_EVENT_GET_USER_COUNT

Description:

The SDK module responds to the user count request stored on the application's terminal.

Notify the application of the event.

UCSAPI_CALLBACK_EVENT_GET_USER_INFO_LIST

Description:

The SDK module responds to requests for the user list stored on the application's terminal.

Notify the event to the application.

UCSAPI_CALLBACK_EVENT_GET_USER_DATA

Description:

The SDK module responds to requests for user data stored on the application's terminal.

Notify the event to the application.

UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO

Description:

The terminal acquires the information required for authentication (Authentication Type) during 1:1 server authentication by using the application program.

The application notifies the terminal of this event. At this time, the application sends the user's authentication type information to the terminal.

It must be transmitted. This event occurs only when authentication is performed on the server and always precedes the following event.

will be published.

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1,

UCSAPI_CALLBACK_EVENT_VERIFY_CARD,

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1

Description:

The terminal notifies the application of this event during 1:1 fingerprint authentication. At this time, the application

The authentication result must be sent to the terminal. This event occurs only when authentication is performed on the server.

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N

Prototype:

```
#define UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N UCSAPI_CALLBACK_EVENT+22
```

Description:

The terminal notifies the application of this event during 1:N fingerprint authentication. At this time, the application

The authentication result must be sent to the terminal. This event occurs only when authentication is performed on the server.

UCSAPI_CALLBACK_EVENT_VERIFY_CARD

Description:

The terminal notifies the application of this event during card authentication. At this time, the application authenticates

The results must be transmitted to the terminal. This event occurs only when authentication is performed on the server.

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD

Description:

The terminal notifies the application of this event during password authentication. At this time, the application

The authentication result must be transmitted to the terminal. This event occurs only when authentication is performed on the server.

UCSAPI_CALLBACK_EVENT_GET_TERMINAL_OPTION

Description:

The SDK module responds to the application's request for terminal option settings by sending this event to the application.

Notify via program.

UCSAPI_CALLBACK_EVENT_SET_TERMINAL_OPTION

Description:

The SDK module responds to the application's request for terminal option settings by sending this event to the application.

Notify via RAM.

UCSAPI_CALLBACK_EVENT_FW_UPGRADING

Description:

The SDK module responds to the application's firmware upgrade request by triggering this event in the application.

Notify in grams. This is an event to notify the application of the progress of the upgrade.

UCSAPI_CALLBACK_EVENT_FW_UPGRADED

Description:

The SDK module responds to the application's firmware upgrade request by confirming the upgrade completion.

Notify via the Yong program.

UCSAPI_CALLBACK_EVENT_FW_VERSION

Description:

The SDK module sends this event to the application in response to the application's firmware version request.

We hereby notify you.

UCSAPI_CALLBACK_EVENT_USERFILE_UPGRADING

Description:

The SDK module responds to this event in the application in response to the user's file download request.

Notify via the application. An event to notify the application of the download progress.

d.

UCSAPI_CALLBACK_EVENT_USERFILE_UPGRADED

Description:

The SDK module responds to the application's user file download request by acknowledging the download completion.

Notify via the Yong Program.

UCSAPI_CALLBACK_EVENT_OPEN_DOOR

Description:

The SDK module responds to the application's request to temporarily open the door by triggering this event in the application.

Notify by gram.

UCSAPI_CALLBACK_EVENT_TERMINAL_CONTROL

Description:

The SDK module transmits this event to the application in response to the application's terminal control.

It is underground.

.

UCSAPI_CALLBACK_EVENT_PICTURE_LOG

Description:

When the SDK module receives a Picture log from the terminal, it processes the Picture log data within the application. by Notice does. terminal Certification by means of Certification Han Gyeong-won Picture The log is The UCSAPI_CALLBACK_EVENT_REALTIME_ACCESS_LOG event is not notified to the application. When authenticating using the server authentication method, the terminal receives the authentication result from the application and then Picture logs are notified separately as an application.

UCSAPI_CALLBACK_EVENT_ANTIPASSBACK

Description:

If the terminal has the Antipassback option enabled, it will use the user's current Notify the application of this event to obtain the antipassback status. At this time, the application... The RAM must immediately transmit the user's Antipassback status to the terminal. This event must be transmitted by the terminal to the terminal. This occurs only when performing authentication on the server.

UCSAPI_CALLBACK_EVENT_SET_ACCESS_CONTROL_DATA

Description:

The SDK module responds to the application's request for device access permissions by triggering this event. Notify via the program.

UCSAPI_CALLBACK_EVENT_REGISTER_FACE

Description:

The SDK module responds to the application's face registration via the terminal with this event. Notify by gram. The terminal displays "Face registration in progress" on the LCD screen in response to a face registration request and begins capturing the face. Face registration requires 5 photos for standard registration and 3 photos for simplified registration. Each step of this event will be launched. It is executed. When registration is canceled on the terminal, an event for the cancellation is issued, but the server-side requests the cancellation. In one case, the registration process ends when it is canceled on the terminal.

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0_PTR

The CurrentIndex value of Progress increases incrementally starting from 1 based on the progress status. When the TotalNumber and CurrentIndex values of Progress are 0, it indicates that registration has been canceled.

lParam:

UCSAPI_CALLBACK_PARAM_1_PTR

- DataType: UCSAPI_CALLBACK_DATA_TYPE_FACE_INFO

- Data: UCSAPI_FACE_INFO_PTR

Upon normal face input, the address of the face data structure is received.

UCSAPI_CALLBACK_EVENT_GET_ACU_OPTION

Description:

The SDK module is a response to the application's request to obtain the ACU option value.

Callback Parameters:

wParam: Result value

UCSAPI_CALLBACK_PARAM_0 pCallback0;

lParam: Address of the structure containing the requested option value

UCSAPI_ACU_OPTION_PTR pOption;

UCSAPI_CALLBACK_EVENT_SET_ACU_OPTION

Description:

The SDK module is a response to the application's terminal option value settings.

Callback Parameters:

wParam: Result value

UCSAPI_CALLBACK_PARAM_0 pCallback0;

lParam: Not used

UCSAPI_CALLBACK_EVENT_GET_ACU_LOCK_SCHEDULE

Description:

This is the response for obtaining the Lock Schedule value for the specified ACU's Lock.

Callback Parameters:

wParam: Result value

UCSAPI_CALLBACK_PARAM_0 pCallback0;

lParam: Address of the structure containing the requested Schedule value

UCSAPI_ACU_LOCK_SCHEDULE_PTR pSchedule;

UCSAPI_CALLBACK_EVENT_SET_ACU_LOCK_SCHEDULE

Description:

This is the response to the Lock Schedule value setting for the specified ACU's Lock.

Callback Parameters:

wParam: Result value

UCSAPI_CALLBACK_PARAM_0 pCallback0;

IParam: Not used

UCSAPI_CALLBACK_EVENT_GET_SIREN

Description:

This is the response to the result of executing UCSAPI_SendSirenToTerminal. (When the opt value is 0)

Callback Parameters:

wParam: Result value

UCSAPI_CALLBACK_PARAM_0 pCallback0;

When the opt value is 0, pCallback0->Progress.TotalNumber represents the number of subsequent siren data points.

|Param:

When the opt value is 0, it has the UCSAPI SIREN CONFIG PTR value.

UCSAPI_CALLBACK_EVENT_SET_SIREN

Description:

This is the response to the result of executing UCSAPI SendSirenToTerminal. (When the opt value is 1)

Callback Parameters:

wParam: Result value

UCSAPI_CALLBACK_PARM_0 pCallback0;

|Param: Not used

UCSAPI CALLBACK EVENT SET SMARTCARD LAYOUT

Description:

This is the response to the result of executing UCSAPI_SetSmartCardLayoutToTerminal.

Callback Parameters:

wParam: Result value

UICSAPI_CALLBACK_PARAM_0 **nCallback:**

Baron: Nature

UICSAPI CALLBACK EVENT FP MINUTIAE

Description:

This is the response to the result of executing UICSAPI_GetEnMinutiaeFromTerminal

Callback Parameters:

wParam: Result value

```
UCSAPI_CALLBACK_PARAM_0           pCallback0;  
/Param: Acquired fingerprint feature point information  
UCSAPI_FP_MINUTIAE_PTR          pMinutiae;
```

3.4 Error Definitions

This document defines and explains the various error values used in the UCS SDK.

All error values are defined in the UCSAPI_Error.h file.

3.4.1 Success

This is the definition of the Error value used upon success.

UCSAPIERR_NONE

Prototype:

```
#define UCSAPIERR_NONE           (0)
```

Description:

The value returned when the function succeeds. In this case, it indicates the function succeeded, not an error.

3.4.2 General error definitions

This is the definition of common error values.

This error value starts with the value UCSAPIERR_BASE_GENERAL (0).

UCSAPIERR_INVALID_POINTER

Prototype:

```
#define UCSAPIERR_INVALID_POINTER    (0x0001)
```

Description:

An invalid pointer value was used.

UCSAPIERR_INVALID_TYPE

Prototype:

```
#define UCSAPIERR_INVALID_TYPE (0x0002)
```

Description:

An incorrect Type value is being used.

UCSAPIERR_INVALID_PARAMETER

Prototype:

```
#define UCSAPIERR_INVALID_PARAMETER (0x0003)
```

Description:

An incorrect parameter value was used.

UCSAPIERR_INVALID_DATA

Prototype:

```
#define UCSAPIERR_INVALID_DATA (0x0004)
```

Description:

Incorrect data values were used.

UCSAPIERR_FUNCTION_FAIL

Prototype:

```
#define UCSAPIERR_FUNCTION_FAIL (0x0005)
```

Description:

Occurs when a function execution fails due to an internal function error.

UCSAPIERR_NOT_SERVER_ACTIVE

Prototype:

```
#define UCSAPIERR_NOT_SERVER_ACTIVE (0x0006)
```

Description:

The server is not in the initialized state.

UCSAPIERR_INVALID_TERMINAL

Prototype:

```
#define UCSAPIERR_INVALID_TERMINAL (0x0007)
```

Description:

The device is not connected.

UCSAPIERR_PROCESS_FAIL

Prototype:

```
#define UCSAPIERR_PROCESS_FAIL (0x0008)
```

Description:

Failure during processing.

UCSAPIERR_USER_CANCEL

Prototype:

```
#define UCSAPIERR_USER_CANCEL (0x0009)
```

Description:

Undo by user.

UCSAPIERR_UNKNOWN_REASON

Prototype:

```
#define UCSAPIERR_UNKNOWN_REASON (0x0010)
```

Description:

Unknown error.

3.4.3 Data size-related error definitions

This is the definition of the data size error value.

This error value starts with the UCSAPIERR_BASE_MEMORY (0x0200) value.

UCS API ERROR CODE SIZE

Prototype:

```
#define UCSAPIERR_CODE_SIZE (0x0201)
```

Description:

Access Group Code value size exceeded.

UCSAPIERR_USER_ID_SIZE

Prototype:

```
#define UCSAPIERR_USER_ID_SIZE (0x0202)
```

Description:

Maximum size of user ID value exceeded.

UCSAPIERR_USER_NAME_SIZE

Prototype:

```
#define UCSAPIERR_USER_NAME_SIZE (0x0203)
```

Description:

Maximum size exceeded for the username value.

UCSAPIERR_UNIQUE_ID_SIZE

Prototype:

```
#define UCSAPIERR_UNIQUE_ID_SIZE (0x0204)
```

Description:

Maximum size of UNIQUE ID value exceeded.

UCSAPIERR_INVALID_SECURITY_LEVEL

Prototype:

```
#define UCSAPIERR_INVALID_SECURITY_LEVEL (0x0205)
```

Description:

Certification level value exceeds the range.

UCSAPIERR_PASSWORD_SIZE

Prototype:

```
#define UCSAPIERR_PASSWORD_SIZE (0x0206)
```

Description:

Maximum size of user password value exceeded.

UCSAPIERR_PICTURE_SIZE

Prototype:

```
#define UCSAPIERR_PICTURE_SIZE (0x0207)
```

Description:

Maximum size exceeded for user photo image value.

UCSAPIERR_INVALID_PICTURE_TYPE

Prototype:

```
#define UCSAPIERR_INVALID_PICTURE_TYPE (0x0208)
```

Description:

The user photo image type is not supported.

UCSAPIERR_RFID_SIZE

Prototype:

```
#define UCSAPIERR_RFID_SIZE (0x0209)
```

Description:

Maximum size of user card number value exceeded.

UCSAPIERR_MAX_CARD_NUMBER

Prototype:

```
#define UCSAPIERR_MAX_CARD_NUMBER (0x0211)
```

Description:

The maximum number of cards that can be registered has been exceeded. Each user can register up to 5 cards.

UCSAPIERR_MAX_FINGER_NUMBER

Prototype:

```
#define UCSAPIERR_MAX_FINGER_NUMBER (0x0212)
```

Description:

The maximum number of fingerprints that can be registered has been exceeded. Up to 5 fingerprints can be registered per user.

3.4.4 Authentication-related error definitions

This is the definition of error values related to authentication.

This error value starts with the UCSAPIERR_BASE_AUTHENTICATION (0x0300) value.

UCSAPIERR_INVALID_USER

Prototype:

```
#define UCSAPIERR_INVALID_USER (0x0301)
```

Description:

Unregistered user.

UCSAPIERR_UNAUTHORIZED

Prototype:

```
#define UCSAPIERR_UNAUTHORIZED (0x0302)
```

Description:

Fingerprint, card, and password matching failed.

UCSAPIERR_PERMISSION

Prototype:

```
#define UCSAPIERR_PERMISSION (0x0303)
```

Description:

No authentication privileges.

UCSAPIERR_FINGER_CAPTURE_FAIL

Prototype:

```
#define UCSAPIERR_FINGER_CAPTURE_FAIL (0x0304)
```

Description:

Fingerprint capture failed.

UCSAPIERR_DUP_AUTHENTICATION

Prototype:

```
#define UCSAPIERR_DUP_AUTHENTICATION (0x0305)
```

Description:

Consecutive authentication attempts. Used to prevent dual authentication on the card when operating in drinking water mode.

UCSAPIERR_ANTIPASSBACK

Prototype:

```
#define UCSAPIERR_ANTIPASSBACK (0x0306)
```

Description:

Anti-passback authentication failed.

UCSAPIERR_NETWORK

Prototype:

```
#define UCSAPIERR_NETWORK (0x0307)
```

Description:

No response from the server due to network issues.

UCSAPIERR_SERVER_BUSY

Prototype:

```
#define UCSAPIERR_SERVER_BUSY (0x0308)
```

Description:

The server is busy and cannot perform authentication.

UCSAPIERR_FACE_DETECTION

Prototype:

```
#define UCSAPIERR_FACE_DETECTION (0x0309)
```

Description:

Face authentication failed when using the face recognition feature.

4. API Reference for COM

This chapter describes the properties and methods for using the COM module UCSAPICOM.dll.

4.1 UCSAPI Object

UCSAPICOM.dll serves as the main object for utilizing the UCS SDK, providing its core functionality and supporting various child objects.

It functions as the base Object that retrieves the data. Therefore, to use the SDK, this Object must be declared.

You must.

4.1.1 Properties

This document describes the various properties of the UCSAPI Object.

ErrorCode

Prototype:
[ReadOnly] long ErrorCode;

Description:

Contains values for errors that occurred during the execution of methods and property settings.

A value of 0 indicates success, while any other value indicates failure.

Errors occurring during the configuration of a child object's methods and properties can also be obtained using this value.

EventError

Prototype:
[ReadOnly] long EventError

Description:

When an event is received, it contains the error value for that event.

A value of 0 indicates success, while any other value indicates failure.

Terminal Connections

Prototype:

[ReadOnly] long ConnectionsOfTerminal;

Description:

It contains the value for the number of terminals connected to the server.

This value can be obtained after calling the GetTerminalCount() method.

Terminal User Data

Prototype:

[ReadOnly] VARIANT Terminal UserData;

Description:

Obtain an interface to retrieve user information from the terminal.

GetUserInfoListFromTerminal / GetUserDataFromTerminal Method Call 후

EventGetUserInfoList / EventGetUserData callback event to obtain user information

You must obtain and use the face. For details, refer to the IterminalUserData documentation.

Server User Data

Prototype:

[ReadOnly] VARIANT ServerUserData;

Description:

Obtain an interface for transmitting user information to the terminal.

Before calling the AddUserToTerminal method, configure the user information to be sent to the terminal.

You must obtain and use this interface. For details, refer to the IServerUserData documentation.

AccessLogData

Prototype:

[ReadOnly] VARIANT AccessLogData;

Description:

Obtain an interface to retrieve authentication logs from the terminal.
After calling the GetAccessLog method, the EventGetAccessLog callback event occurs for authentication.
To obtain it, you must acquire and use this interface. For details, refer to the IAccessLogData specification.
For reference.

AccessControlData

Prototype:
[ReadOnly] VARIANT AccessControlData;

Description:
Obtain an interface for configuring access control data via the terminal.
SetAccessControlDataToTerminal Method To set access control data before calling this interface
You must obtain and use an IAccessControlData. For details, refer to the IAccessControlData documentation.

Terminal MAC Address

Prototype:
[ReadOnly] LONG TerminalID
[ReadOnly] BSTR pVal;

Description:
Obtain the MAC address of the specified terminal. (The MAC address is transmitted as a hex string value.)

SampleNumber

Prototype:
[ReadOnly] long SamplesPerFinger;

Description:
It contains the number of templates for the last enrolled FIR.
The value of 1 or 2 will be stored.
This value can be obtained after calling the EnrollFromTerminal() method.

Total Finger Count

Prototype:
[ReadOnly] long FingerNum;

Description:

It holds the number of fingers for the last enrolled FIR.

Values from 1 to 10 will be stored.

This value can be obtained after calling the EnrollFromTerminal() method.

Initialize Packet Encryption

Prototype:

[WriteOnly] bool PacketCrypto;

Description:

Set whether the packet is encrypted.

0: not use, 1: use (default)

InitializeAlpetasSync

Prototype:

[WriteOnly] bool AlpetasSync;

Description:

Configure AlpetasSync availability.

0: not use, 1: use (default)

Terminal Authentication Type: Iris

Prototype:

[ReadOnly] long Iris;

Description:

Retrieves whether the connected device supports iris authentication.

0: not supported, 1: supported

Terminal Authentication Type: Password

Prototype:

[ReadOnly] long Password;

Description:

Retrieves whether the connected device supports password authentication.

0: not supported, 1: supported

Terminal Authentication Type Face

Prototype:

[ReadOnly] long Face;

Description:

Retrieves whether the connected device supports Face authentication.

0: not supported, 1: supported

Terminal Authentication Type Card

Prototype:

[ReadOnly] long Card;

Description:

Retrieves whether the connected device supports Card authentication.

0: not supported, 1: supported

Terminal Authentication Type Fingerprint

Prototype:

[ReadOnly] long Finger;

Description:

Retrieves whether fingerprint authentication is supported on the connected device.

0: not supported, 1: supported

Terminal Authentication Type WT Face

Prototype:

[ReadOnly] long WTFace;

Description:

Retrieves whether the connected device supports Walk Through Face authentication.

0: not supported, 1: supported

4.1.2 Methods

This document describes the various methods of the UCSAPI Object.

ServerStart

Prototype:

HRESULT ServerStart(long MaxTerminal, long Port);

Description:

Initializes the UCSAPI SDK and executes server functions.

Parameters:

MaxTerminal:

Defines the maximum number of connected terminals. The SDK preemptively allocates internal memory usage based on the entered maximum terminal count.

You can improve speed by allocating resources. If the number of connected devices exceeds the maximum, memory usage will automatically increase.

Secure the number of connections.

Port:

Communication port for terminal connection. Uses 9870 as the default value. If this value is changed, the terminal's port...

The value of t must also be changed.

Related Properties

ErrorCode

Callback Event:

EventTerminalConnected(long TerminalID, BSTR TerminalIP);

Event Parameters:

TerminalID:

Device ID

TerminalIP:

A string containing the IP address value of the terminal.

ServerStop

Prototype:

```
HRESULT ServerStop();
```

Description:

Disconnect all connected terminals and terminate server functions.

Parameters:

N/A

Related Properties

ErrorCode

Callback Event:

N/A

Event Parameters

N/A

SetTerminalTime

Prototype:

HRESULT SetTerminalTime(SHORT year, BYTE mon, BYTE day, BYTE hour, BYTE min, BYTE sec)

Description:

When a terminal requests the current time, COM raises an event called EventGetTerminalTime.

It is set. At this time, when the current time is set via the corresponding method in the API, that time is sent to the terminal.

The terminal's current time is set upon transmission. If there is no processing for the EventGetTerminalTime event,

COM sets the terminal's current time by treating the system time as the current time.

Parameters:

year, month, day, hour, minute, second:

The current date and time values to be set

Related Properties

N/A

Callback Event:

N/A

Event Parameters

N/A

SetTerminalTimezone

Prototype:

```
HRESULT SetTerminalTimezone(long TerminalID, BSTR TimezoneName);
```

Description:

Set the time zone for the connected device.

The terminal generally uses the time zone of the connected server. However, the terminal's time zone and

If the server's time zone is incorrect, the terminal's time zone must be set to the local time zone.

Parameters:

TerminalID:

Device ID

TimezoneName:

Specify the name of the time zone to change.

The time zone name can be found under the following registry entry.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\TimeZones

Related Properties

ErrorCode

Callback Event:

N/A

Event Parameters

N/A

※ GMT stands for

When referring to standard time zones, GMT is generally used as a reference.

GMT stands for Greenwich Mean Time, with Greenwich, England serving as the starting point for standard time.

is used.

For example, Korea is GMT+9, which means it is 9 hours ahead of Greenwich Mean Time.

The place with the earliest time is Wellington, New Zealand. It is 3 hours ahead of Korea, so the time in Wellington is...

It is 10:00 AM.

New York is GMT-5, so here it's 5 p.m. the previous day.

※ Time zone Name

The time zone name can be referenced from the following registry entry.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\TimeZones

	이름	종류	데이터
	(기본값)	REG_SZ	(값 설정 안됨)
	Display	REG_SZ	(GMT-09:00) 알래스카
	Dlt	REG_SZ	알래스카 일광 절약 시간제
	Index	REG_DWORD	0x00000003 (3)
	MapID	REG_SZ	30.31
	Std	REG_SZ	알래스카 표준시
	TZI	REG_BINARY	1c 02 00 00 00 00 00 c4 ff ff ff 00 00 0b 00

SetError

Prototype:

HRESULT SetError(long TerminalID, long EventType);

Description:

Configured to return an error code to the terminal for Callback Events received from the terminal.

d.

Parameters:

TerminalID:

Device ID

EventType:

It holds the Error Type value for returning to the terminal.

. 0 – None

1 – Access Log

For example, after calling GetAccessLogFromTerminal, store the received log data in the database.

If an error occurs, setting EventType to 1 within the Callback Event function and calling SetError will cause the terminal to return to the previous state.

Keep the log in a new log state so that when the next New Log request is made,

I will send it again.

Related Properties

ErrorCode

Related Callback Event:

Get Access Log from Terminal

Event Parameters

GetTerminalCount

Prototype:

HRESULT GetTerminalCount(void);

Description:

Retrieve the number of terminals connected to the server. Use the ConnectionsOfTerminal property to obtain it.

It is possible.

Parameters:

N/A

Related Properties

ErrorCode, Terminal Connections

Callback Event:

N/A

Event Parameters

N/A

GetFirmwareVersionFromTerminal

Prototype:

```
HRESULT GetFirmwareVersionFromTerminal(long ClientID, long TerminalID);
```

Description:

Retrieves the firmware version of the specified device.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID

Related Properties

ErrorCode

Callback Event:

```
EventFirmwareVersion(long ClientID, long TerminalID, BSTR Version);
```

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

Version:

It contains the firmware version value as a string.

Upgrade Firmware to Terminal

Prototype:

```
HRESULT UpgradeFirmwareToTerminal(long ClientID, long TerminalID, BSTR FilePath);
```

Description:

Upgrade the firmware of the specified device. Upgrade progress information is notified via the following events.

Event Firmware Upgrading / Event Firmware Upgraded

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

FilePath:

Specify the full path of the firmware file as a string.

Related Properties

ErrorCode

Callback Event:

```
EventFirmwareUpgrading(long ClientID, long TerminalID, long CurrentIndex, long TotalNumber);
```

```
EventFirmwareUpgraded(long ClientID, long TerminalID);
```

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

CurrentIndex:

It contains the index value of the data block currently being transmitted.

TotalNumber:

It contains the total number of data blocks to be transmitted.

SendUserFileToTerminal

Prototype:

HRESULT SendUserFileToTerminal(long ClientID, long TerminalID, LONG FileType, BSTR FilePath);

Description:

Download user files to the specified device.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

FileType:

Specify the type of file to download. (See UCSAPI_SendUserFileToTerminal)

FilePath:

Specify the absolute path of the file.

Related Properties

ErrorCode

Callback Event:

EventUserFileUpgrading(long ClientID, long TerminalID, long CurrentIndex, long TotalNumber);

EventUserFileUpgraded(long ClientID, long TerminalID);

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

CurrentIndex:

It contains the index value of the data block currently being transmitted.

TotalNumber:

It contains the total number of data blocks to be transmitted.

Open Door to Terminal

Prototype:

HRESULE OpenDoorToTerminal(long ClientID, long TerminalID);

Description:

Temporarily unlocks the specified device.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

Related Properties

ErrorCode

Callback Event:

EventOpenDoor(long ClientID, long TerminalID);

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

SetDoorStatusToTerminal

Prototype:

HRESULE SetDoorStatusToTerminal(long ClientID, long TerminalID, long Status);

Description:

Controls the lock mechanism of the designated terminal according to its status.

Parameters:

[ClientID: 1.6 See term explanation](#)

[TerminalID: 1.6 See Terminology Explanation](#)

[Status:](#)

Lock Status (0: Temporary Open, 1: Lock Released, 2: Lock Corrected)

Related Properties

ErrorCode

Callback Event:

Event Open Door

SendTerminalControl

Prototype:

HRESULE SendTerminalControl(long ClientID, long TerminalID, long lockStatus, long lockType)

Description:

Control the specified terminal according to the param value.

Parameters:

[ClientID:](#)

The client ID of the client who requested the task.

[TerminalID:](#)

Device ID.

[lockStatus:](#)

Lock Status (0: Unlock, 1: Lock)

[lockType:](#)

Lock Type (0:Normal, 1:Global)

Related Properties

ErrorCode

Callback Event:

EventTerminalControl (long ClientID, long TerminalID, long lockStatus, long lockType);

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

lockStatus:

Lock Status (0: Unlock, 1: Lock)

lockType:

Lock Type (0:Normal, 1:Global)

SendPrivateMessageToTerminal

Prototype:

```
HRESULT SendPrivateMessageToTerminal(  
    LONG ClientID, LONG TerminalID,  
    LONG Reserved, BSTR TextMessage, short displayTime)
```

Description:

Display the specified text on the LCD screen of the designated terminal for a specified duration.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

TextMessage:

Specify the text to display. (See UCSAPI_SendPrivateMessageToTerminal)

displayTime :

Specify the time to display on the screen in seconds.

Related Properties

Callback Event:

EventPrivateMessage(long ClientID, long TerminalID);

Event Parameters

[ClientID: 1.6 See term explanation](#)

[TerminalID: 1.6 See Terminology Explanation](#)

SendPublicMessageToTerminal

Prototype:

```
HRESULT SendPublicMessageToTerminal(  
    LONG ClientID, LONG TerminalID, BOOL Show,  
    BSTR StartDate, BSTR EndDate, BSTR StartTime, BSTR EndTime, BSTR TextMessage)
```

Description:

Display the notice on the LCD screen of the designated terminal for the specified period and at the specified time.

However, if the Show value is FALSE, the notice is not displayed.

Parameters:

[ClientID: 1.6 See term explanation](#)

[TerminalID: 1.6 See Terminology Explanation](#)

[TextMessage:](#)

Specify the message to display. (See UCSAPI_SendPublicMessageToTerminal)

[StartDate, EndDate :](#)

Specify the period to display on the screen in date format (e.g., "2012-03-25" or "20120325")

[StartTime, EndTime :](#)

Specify the time to display on the screen in hours and minutes (e.g., "08:30" or "0830")

Related Properties

Callback Event:

EventPublicMessage(long ClientID, long TerminalID);

Event Parameters

[ClientID: 1.6 See term explanation](#)

[TerminalID: 1.6 See Terminology Explanation](#)

Set Wiegand Format to Terminal

Prototype:

```
HRESULT SetWiegandFormatToTerminal(LONG ClientID, LONG TerminalID, LONG wgdType,  
BSTR FilePath);
```

Description:

Configure the terminal's Wiegand In/Out Format. The Wiegand Format File is provided with the SDK.

It can be generated using a Wiegand Tool.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

wgdType:

It has a Type value to distinguish between Wiegand In and Wiegand Out.

FilePath:

Full path to the Wiegand Format Data File.

Related Properties

ErrorCode

Related Callback Event:

Event Set Wiegand Format

Event Parameters

`SetDoorStatusToACU`

Prototype:

`HRESULE SetDoorStatusToTerminal(long ClientID, long TerminalID, long Status, long DoorID);`

Description:

Controls the lock mechanism for the designated ACU door according to its status.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Status:

Lock Status (0: Temporary Open, 1: Lock Release, 2: Lock Correction, 3: Set Configuration, 4: Set Cancellation)

DoorID:

Door ID value of the ACU to be controlled

Related Properties

ErrorCode

Callback Event:

Event Open Door

Event ACU Status

`GetFpMinutiaeFromTerminal`

Prototype:

`HRESULE GetFpMinutiaeFromTerminal (long ClientID, long TerminalID, BYTE minType, BYTE minCount);`

Description:

The terminal acquires fingerprint data to obtain feature point information.

For more details, please refer to UCSAPI_GetFpMinutiaeFromTerminal.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

minType

Specifies the type of feature point information to extract (currently 0: only union types are supported)

minCount

Specify the number of fingerprints to be entered. (Default: 2 entries)

The terminal checks whether the fingerprint received is from the same finger through matching.

Related Properties

ErrorCode

Callback Event:

GetMinutiaeFromTerminal

SetEmergencyToTerminal

Prototype:

```
HRESULE SetEmergencyToTerminal(long ClientID, long TerminalID, long SignalType,  
long SignalValue, long DoorControl, long AlarmControl, long Reserved1, long Reserved2)
```

Description:

Configure the emergency-related settings for the designated terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

SignalType:

1:FIRE, 2: PANIC

SignalValue:

0:STOP, 1:START

DoorControl:

0:None, 1:Open

Alarm Control:

0:None, 1:Play

Related Properties

ErrorCode

Callback Event:

EventSetEmergency(long ClientID, long TerminalID);

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

Terminal Options Dialog

Prototype:

HRESULE TerminalOptionDialog(long ClientID, long TerminalID)

Description:

Displays a dialog providing terminal settings-related functions for the specified terminal.

(Windows platform only)

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related Properties

ErrorCode

Callback Event:

```
EventGetTerminalOption(long ClientID, long TerminalID);
```

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

OpenDoorToTerminalEx

Prototype:

```
HRESULE OpenDoorToTerminalEx(long ClientID, long TerminalID, long OpenTime)
```

Description:

This is an extended API for the OpenDoorToTerminal function. It relates to the door open duration.

A parameter has been added.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

OpenTime:

Door open duration.

Related Properties

ErrorCode

Callback Event:

```
EventOpenDoor(long ClientID, long TerminalID);
```

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

GetUserInfoListFromTerminal

Prototype:

HRESULE GetUserInfoListFromTerminal(long ClientID, long TerminalID)

Description:

Retrieves the list of all registered user information from the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related Properties

ErrorCode

Callback Event:

EventGetUserInfoList (long ClientID, long TerminalID);

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

Control Peripheral Device

Prototype:

HRESULE ControlPeripheralDevice(long ClientID, long TerminalID, long PeripheralDeviceID,
long ControlCommand)

Description:

Control the peripheral devices of the designated terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

PeripheralDeviceID:

Peripheral device ID.

ControlCommand:

Command settings

Related Properties

ErrorCode

Callback Event:

```
EventControlPeripheralDevice(long ClientID, long TerminalID, long PeripheralDeviceID,  
long ControlCommand);
```

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

get_FingerID

Prototype:

```
HRESULE get_FingerID(long nIndex)
```

Description:

Finally, it retrieves the finger ID of the index specified in the last enrolled FIR.

Parameters:

nIndex:

FIR index.

Related Properties

ErrorCode, FingerID

get_FPSampleData

Prototype:

HRESULE get_FPSampleDataID(int nFingerID, int nSampleNum)

Description:

Finally, it retrieves the template data for the finger ID specified in the last enrolled FIR.

Parameters:

nFingerID:

Finger ID.

nSampleNum:

Template number.

Each Finger ID has two sets of template data.

Related Properties

ErrorCode, Template data

EnrollFromTerminal

Prototype:

HRESULE EnrollFromTerminal(int ClientID, int TerminalID)

Description:

When this command is issued, a fingerprint will be requested from the designated terminal.

The fingerprint will be scanned twice until the stop command is issued.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related Properties

ErrorCode

get_FPSampleDataLength

Prototype:

HRESULE get_FPSampleDataLength(int nFingerID, int nSampleNum)

Description:

Finally, it retrieves the template data size of the finger ID specified in the last enrolled FIR.

Parameters:

nFingerID:

The client ID of the client who requested the task.

nSampleNum:

Template number.

Each Finger ID has two sets of template data.

Related Properties

Error Code, Template Data Size

Reboot to Terminal

Includes note [w3]: Added on 20241227

Prototype:

```
HRESULT RebootToTerminal(long ClientID, long TerminalID);
```

Description:

Force a reboot of the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related Properties

ErrorCode, Template data size

Callback Event:

```
EventRebootTerminal(long ClientID, long TerminalID);
```

Event Parameters

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID

4.2 IServerUserData Interface

Interface for transmitting user information to the terminal. User information for transmitting to the terminal.

After setting the relevant properties, call the AddUserToTerminal method. After sending the user information,

Check the error code of the EventAdduserToTerminal event to verify that the transmission was completed successfully.

4.2.1 Properties

This section describes the various properties of the IServerUserData interface.

UserID

Prototype:

[WriteOnly] long UserID;

Description:

Specify the value for the user ID. This value can only be specified as numeric data with a maximum of 8 digits.

UniqueID

Prototype:

[WriteOnly] BSTR UniqueID;

Description:

Specify the unique number (employee number) value as a string. This value is used instead of UserID for user identification.

It is possible. The maximum data size that can be specified is up to 20 bytes.

UserName

Prototype:

[WriteOnly] BSTR UserName;

Description:

Specify the user name value as a string. The maximum data length that can be specified is 16 bytes.

AccessGroup

Prototype:
[WriteOnly] BSTR AccessGroup;

Description:

Specify the access group code value as a string. The code value consists of a fixed 4-byte string.

SecurityLevel

Prototype:
[WriteOnly] long SecurityLevel;

Description:

You can set the authentication security level to be used for fingerprint authentication. This value can be set to the following values:
can have.

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

By default, 1:1 has a value of 4, and 1:N has a value of 5.

IsCheckSimilarFinger

Prototype:
[WriteOnly] BOOL IsCheckSimilarFinger;

Description:

Specifies whether to check for similar fingerprints when adding the user's fingerprint data to the device.

If this value is set to True, the terminal compares the fingerprint with those of all registered users to detect a similar fingerprint.

If a duplicate fingerprint is detected during the existence check, the registration is treated as a failure. This flag is used as a terminal.

Additionally, slowing down additional processing becomes a factor that degrades performance when there are many fingerprint registration users.
d.

IsAdmin

Prototype:
[WriteOnly] BOOL IsAdmin;

Description:

Users can be designated as device administrators. Devices with one or more registered administrators have settings
Accessing the menu requires administrator login, which can restrict terminal menu usage.

IsIdentify

Prototype:
[WriteOnly] BOOL IsIdentify;
Description:
Users can be designated to enable 1:N fingerprint authentication.

Password

Prototype:
[WriteOnly] BSTR Password;

Description:
When the user employs a password authentication method, they can specify the password string. Maximum
The maximum data size that can be specified is 8 bytes.

FaceNumber

Prototype:
[WriteOnly] long FaceNumber;

Description:
This data is included when user information is downloaded to the terminal and is used for facial authentication.
The registered face data (FaceData) contains multiple facial images, and FaceNumber is
Displays the number of registered faces. FaceData contains a minimum of 3 and a maximum of 10 face entries.

Related methods:

Add User to Terminal

Related properties:

FaceData

FaceData

Prototype:
[WriteOnly] long FaceData;

Description:

User facial feature point data downloaded to the terminal device, used during facial authentication.
The number of facial landmarks contained within is specified in FaceNumber.
A single facial feature point data consists of a length (4 bytes) and the actual data (n bytes).

Related methods:

Add User to Terminal

Related properties:

FaceNumber

IsFace1toN

Prototype:
[WriteOnly] long Face1toN;

Description:

Enable 1:N facial recognition for the user.

IsBlacklist

Prototype:
[WriteOnly] long Blacklist;

Description:

Determines whether to blacklist the user.

IsIris1toN

Prototype:
[WriteOnly] long Iris1toN;

Description:

Designate users to enable 1:N iris authentication.

4.2.2 Methods

This document describes the various methods of the IServerUserData interface.

Initialize User Data

Prototype:

HRESULT InitUserData()

Description:

Initialize all variables in the user information.

Parameters:

None

SetAuthenticationType

Prototype:

HRESULT SetAuthType(BOOL AndOperation, BOOL Finger, BOOL FPCard, BOOL Password,
BOOL Card, BOOL CardID);

Description:

You can specify the user's authentication type. Each authentication type is determined based on the AndOperation Flag.

It can be used in combination with AND or OR.

Parameters:

AndOperation:

You can specify that each authentication type be used in combination with others via AND or OR.

Here, set 1 for AND combinations and 0 for OR combinations. For details, refer to Chapter 1.5: Terminology.

Refer to the user attributes.

Finger:

Fingerprint authentication can be enabled.

FPCard:

Fingerprint card authentication can be enabled. Fingerprint cards store fingerprint information on smart cards.

This is a method of certification.

Password:

Password authentication can be enabled.

Card:

Card authentication can be enabled.

CardID:

RFID can be designated for use as a UserID or UniqueID. The CardID corresponds to the RFID on the card.

This refers to using it solely as an identifier, like a UserID, rather than as an authentication method.

It must be specified in combination with other authentication types using an AND operator.

Related methods

AddUserToTerminal, SendAuthenticationInformationToTerminal

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetFPSampleData

- Replaced with AddFingerData.

Prototype:

```
HRESULT SetFPSampleData(BOOL bInitialize, long nSrcFPDataSize,  
VARIANT FPData1, VARIANT FPData2);
```

Description:

Specify the binary stream data of the finger-specific template from the transformed FIR for fingerprint authentication.

Available. For detailed information on the Template, refer to the UCBioBSP SDK.

Parameters:

bInitialize:

Specify whether to initialize the FIR data and create a new one.

If this value is False, the Template data being added now is internally generated FIR data.

As it continues to be added to the site, a single FIR data set containing multiple Template data points is created.

This value must be set to True to clear all existing FIR filters and create new ones.

nSrcFPData :

The type information for the template you wish to add. For related values, refer to UCBioAPI_TEMPLATE_TYPE.

nFPDataSize :

Size data for the template to be added.

FPData1:

Template data to be added. (Binary stream data)

FPData2:

Second template data for the finger to be added. (Binary stream data)

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

AddFingerData

Prototype:

```
HRESULT AddFingerData(long FingerID, long FPData
```

```
VARIANT FPData1, VARIANT FPData2);
```

Description:

For fingerprint authentication, the finger information and the binary stream of the finger-specific template derived from the transformed FIR

Data can be specified. For detailed information about the Template, refer to the UCBioBSP SDK.

After executing InitUserData(), continue adding finger information for use.

Parameters:

FingerID:

Finger information (1: right thumb, 10: left pinky)

FPDataType :

The type information for the template you wish to add. For related values, refer to UCBioAPI_TEMPLATE_TYPE.

FPData1:

Template data to be added. (Binary stream data)

FPData2:

Second template data for the finger to be added. (Binary stream data)

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

Set Duress Finger

Prototype:

HRESULT SetDurationFinger(long FingerID, long Value)

Description:

Designate the threatening fingerprint among the fingerprint information.

Fingerprints registered as blacklisted fingerprints will be marked as successful during authentication on the terminal, but

The server-side receives the authentication result value as 0x21 (33).

Parameters:

FingerID:

Fingerprint information to designate as a threat fingerprint (1: Right thumb, 10: Left ring finger)

Value:

Threat fingerprint presence (0: Normal fingerprint, 1: Threat fingerprint)

Related properties:

ErrorCode

SetCardData

Prototype:

HRESULT SetCardData(BOOL bInitialize, BSTR RFID);

Description:

RFID data can be specified for card authentication.

Parameters:

bInitialize:

Specify whether to initialize the RFID data and create it anew.

If this value is False, the RFID data being added now will be appended to the internally generated RFID data.

As data continues to be added, multiple RFID data entries are created.

Setting this value to True will erase all existing RFID data and create new data.

RFID:

Specify the RFID value to be added as a string. The maximum data that can be specified is up to 16 bytes.

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetPictureData

Prototype:

```
HRESULT SetPictureData(long PictureDataLength, BSTR PictureDataType, VARIANT  
PictureData);
```

Description:

You can specify the photo data as a binary stream.

Parameters:

PictureDataLength:

Specify the length of the photo data.

PictureDataType:

Specify the type value of the photo data as a string. (Currently supports only the "JPG" format)

Specify the file extension value as a string.

PictureData:

Photo data to be added. (Binary stream data)

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetAccessDate

Prototype:

```
HRESULT SetAccessDate(long AccessDateType, long StartYear, long StartMonth, long  
StartDay, long EndYear, long EndMonth, long EndDay);
```

Description:

You can specify periods when access is permitted or prohibited. Access period data can be set to "Not in use" or "Access".

You can specify a total of three data types: available period, unavailable period, and others.

The value is as follows.

Parameters:

Access Date Type:

You can specify the type of data for the access period. The possible values are as follows.

- 0 - Not in use
- 1 - Specify the valid certification period
- 2 - Specify the period during which authentication is not possible

StartYear / StartMonth / StartDay:

Specify the start date of the access period.

EndYear / EndMonth / EndDay:

Specify the end date of the access period.

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetAccessTime

Includes note [w4]: Added on 20250418

Prototype:

```
HRESULT SetAccessTime(long StartHour, long StartMinute, long EndHour, long EndMinute);
```

Description:

You can specify accessible or inaccessible times. Access time data can be set to "Disabled," "Access,"

You can specify a total of three data types: available period and unavailable period. Data Type

The designation uses the type set in the SetAccessDate method.

Parameters:

StartHour / StartMinute :

Specify the hour and minute of the start date for the access time.

EndHour / EndMinute :

Specify the hour and minute of the end date for access times.

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

Add User to Terminal

Prototype:

```
HRESULT AddUserToTerminal(long ClientID, long TerminalID, BOOL IsOverwrite);
```

Description:

Transfers user information to the designated terminal Before calling the AddUserTerminal method, the user settings

You must create user information using the Properties and Methods related to the Bo.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

IsOverwrite:

If the user is already registered, specify whether to overwrite. The default value is 1.

Related properties:

ErrorCode

Callback Event:

EventAddUser

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

※ Precautions

When using the AddUserToTerminal method to send multiple user data to terminals, you must

After calling the AddUserToTerminal method, check the EventAddUser callback event to verify if it was processed correctly.

After verification, the next user data must be transmitted.

SetAuthenticationTypeEx

Prototype:

```
HRESULT SetAuthTypeEx(int Face, int MobileKey, int Reserved1, int Iris, int Reserved3,  
int Reserved4, int Reserved5, int Reserved6);
```

Description:

This is an extended API for the SetAuthType function. It sets the authentication type to Face, MobileKey, or Iris. was added for this purpose.

Parameters:

Face:

Enable or disable Face Type authentication.

MobileKey:

Enable or disable MobileKey type authentication.

Reserved1 ~ Reserved6:

Parameters for additional support.

Iris:

Enable or disable iris authentication.

Related properties:

N/A

Callback Event:

N/A

Event Parameters:

N/A

SetWalkThroughData

Prototype:

```
HRESULT SetWalkThroughData(int DataType, int DataLength, object Data);
```

Description:

The user can input their walk-through face authentication data.

Parameters:

DataType:

Specify the face data type.

(JPG: 12, TEMPLATE: 13)

DataLength:

Specify the face data size.

Data:

face data.

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetIrisData

Prototype:

```
HRESULT SetIrisData(int DataLength, object Data);
```

Description:

The user's iris authentication data can be entered.

Parameters:

DataLength:

Specify the iris data size.

Data:

Iris data.

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

Reset Duress Finger

Prototype:

HRESULT ResetDuressFinger();

Description:

Reset the blackmail fingerprint from the fingerprint information.

Parameters:

Related properties:

N/A

Callback Event:

N/A

Event Parameters:

N/A

4.3 ITerminalUserData Interface

Interface related to managing and retrieving user information on the terminal. Number of users, user information list, You must obtain and use this interface to retrieve or delete user data.

4.3.1 Properties

This section describes the various properties of the ITerminalUserData interface.

CurrentIndex / TotalNumber

Prototype:

[ReadOnly]	long	CurrentIndex;
[ReadOnly]	long	TotalNumber;

Description:

When obtaining a list of multiple user information entries, it contains the total number of entries in the list and the index of the current record.

d. Can be obtained after calling the GetUserInfoListFromTerminal method.

Related methods:

GetUserInfoListFromTerminal,

UserID

Prototype:

[ReadOnly]	long	UserID;
------------	------	---------

Description:

It contains the user ID value.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

UniqueID

Prototype:

[ReadOnly] BSTR UniqueID;

Description:

It contains the unique number (employee number) value.

GetUserInfoListFromTerminal and GetUserDataFromTerminal methods can be obtained after calling them.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

UserName

Prototype:

[ReadOnly] BSTR UserName;

Description:

It contains the user name value.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

AccessGroup

Prototype:

[ReadOnly] BSTR AccessGroup;

Description:

It contains the access group code value.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

IsAdmin

Prototype:
[ReadOnly] BOOL IsAdmin;

Description:

Contains the flag value indicating whether the user is an administrator.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

IsIdentify

Prototype:
[ReadOnly] BOOL IsIdentify;

Description:

Contains the flag value indicating 1:N fingerprint authentication capability.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber, FPSampleData,
SecurityLevel

Access Date Type

Prototype:
[ReadOnly] long Access Date Type;

Description:

Contains the type value of the access period data. Possible values are as follows.

After calling the GetUserDataFromTerminal method, you can obtain it.

0 - Not in use
1 - Specify the valid certification period

2 - Specify the period during which authentication is not possible

Related methods:

[GetUserDataFromTerminal](#)

Related properties:

Start Access Date, End Access Date, Start Access Time, End Access Time

Includes note[w5]: Added on 20250418

Start Access Date/End Access Date

Prototype:

[ReadOnly] BSTR StartAccessDate;

[ReadOnly] BSTR EndAccessDate;

Description:

The start/end dates of the access period are stored as strings.

The data format is yyyy-MM-dd.

After calling the [GetUserInfoListFromTerminal](#)/[GetUserDataFromTerminal](#) method, you can obtain it.

Related methods:

[GetUserDataFromTerminal](#)

Related Properties:

[Access Date Type](#)

Start Access Time/End Access Time

Includes note [w6]: Added on 20250418

Prototype:

[ReadOnly] BSTR StartAccessTime;

[ReadOnly] BSTR EndAccessTime;

Description:

The start/end date of the access time is stored as a string.

The data format is HH:MM.

After calling the [GetUserInfoListFromTerminal](#)/[GetUserDataFromTerminal](#) method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related Properties:

Access Date Type

SecurityLevel

Prototype:

[ReadOnly] long SecurityLevel;

Description:

It contains the authentication security level value to be used for fingerprint authentication.

You can obtain it after calling the GetUserDataFromTerminal method.

The possible values are as follows.

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

By default, 1:1 has a value of 4, and 1:N has a value of 5.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber, FPSampleData

IsAndOperation

Prototype:

[ReadOnly] BOOL IsAndOperation;

Description:

Contains a flag value indicating the use of an AND/OR combination of authentication types. This value represents the authentication type combination of Fingerprint, Card, and Password.

Enable the use of the mouth in AND or OR combinations. For details, refer to Chapter 1.5, Terminology.

Refer to the properties.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, IsFPCard, IsCard, IsPassword

IsFinger

Prototype:

[ReadOnly] BOOL IsFinger;

Description:

Contains the flag value indicating whether the user's fingerprint authentication is enabled.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber,
FPSampleData, SecurityLevel

IsFPCard

Prototype:

[ReadOnly] BOOL IsFPCard;

Description:

Contains the flag value indicating whether the user's fingerprint card authentication is enabled.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation

IsCard

Prototype:

[ReadOnly] BOOL IsCard;

Description:

Contains the user's Card authentication enabled flag value.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, CardNumber, RFID

IsCardID

Prototype:

[ReadOnly] BOOL IsCardID;

Description:

This flag indicates whether the RFID of the card user is used as an ID for user identification.

CardID does not use the card's RFID as an authentication method; it is used solely as an identifier, similar to UserID.

It must be specified in combination with other authentication types using an AND operator.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation

IsPassword

Prototype:

[ReadOnly] BOOL IsPassword;

Description:

Contains the Flag value indicating password authentication is enabled.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, Password

Password

Prototype:

[ReadOnly] BSTR Password;

Description:

The password value is stored as a string. The maximum data size that can be specified is up to 8 bytes.

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

IsPassword

CardNumber

Prototype:

[ReadOnly] long CardNumber;

Description:

When the user employs the card authentication method, it contains the number of registered RFIDs.

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

IsCard, RFID

RFID

Prototype:

[ReadOnly] BSTR RFID (long nIndex);

Description:

When the user employs the card authentication method, it contains the data value of the registered RFID.

After calling the GetUserDataFromTerminal method, you can obtain it.

Parameters:

nIndex

RFID Index Number to be obtained:

Related methods:

GetUserDataFromTerminal

Related properties:

IsCard, CardNumber

Picture Data Length

Prototype:

[ReadOnly] long PictureDataLength;

Description:

It contains the size value of the photo data.

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

PictureData

PictureData

Prototype:

[ReadOnly] VARIANT PictureData;

Description:

It contains photo data values in binary stream format. The data length value is PictureDataLength.

It is contained within the property.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

Picture Data Length

Total Finger Count

Prototype:

[ReadOnly] long TotalFingerCount;

Description:

Contains the total number of fingers in the transformed FIR.

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, FingerID, FPSampleDataLength, SampleNumber, FPSampleData

FingerID

Prototype:

[ReadOnly] long FingerID(long nIndex);

Description:

The converted FIR's finger ID information is stored in an array.

nIndex can take values from 0 to (TotalFingerCount - 1).

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

[GetUserDataFromTerminal](#)

Related properties:

[IsFinger](#), [TotalFingerCount](#), [FPSampleDataLength](#), [SampleNumber](#), [FPSampleData](#)

SampleNumber

Prototype:

[ReadOnly] long SampleNumber;

Description:

Contains the number of templates per finger for the converted FIR. It holds a value of either 1 or 2.

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

[GetUserDataFromTerminal](#)

Related properties:

[IsFinger](#), [TotalFingerCount](#), [FingerID](#), [FPSampleDataLength](#), [FPSampleData](#)

FPSampleData

Prototype:

[ReadOnly] VARIANT FPSampleData(long nFingerID, long nSampleNum);

Description:

It contains the binary stream data of the finger-specific template for the transformed FIR.

nFingerID and SampleNum can be obtained using the FingerID and SampleNumber properties.

The length value of the binary stream data can be obtained using the FPSampleDataLength property.

After calling the GetUserDataFromTerminal method, you can obtain it.

Parameters:

nFingerID:

Finger ID number to be obtained

nSampleNum:

Sample number obtained. Use a value of 0 or 1.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber

FaceNumber

Prototype:

[ReadOnly] long FaceNumber;

Description:

The number of faces contained in the facial landmark data (minimum value is 3, maximum value is 10)

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

FaceData

FaceData

Prototype:

[ReadOnly] VARIANT FaceData;

Description:

Contains converted facial landmark data. The number of facial landmarks contained within is FaceNumber.

Each facial feature point data is specified as consisting of a length (4 bytes) and the actual data (n bytes).

After calling the GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserDataFromTerminal

Related properties:

FaceNumber

IsBlacklist

Prototype:

[ReadOnly] long IsBlacklist;

Description:

It contains the blacklist flag value for the user.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber, FPSampleData,

SecurityLevel

IsFace

Prototype:

[ReadOnly] long IsFace;

Description:

Contains the flag value indicating whether the user's face authentication is enabled.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, FaceData, FaceNumber, WalkThroughData, WalkThroughLength,
WalkThroughType, WalkThroughData_2, WalkThroughLength_2, WalkThroughType_2

IsIris

Prototype:
[ReadOnly] long IsIris;

Description:

Contains the flag value indicating whether the user's iris authentication is enabled.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, IrisDataLength, IrisData

IsMobileKey

Prototype:
[ReadOnly] long IsMobileKey;

Description:

Contains the user's MobileKey authentication enabled flag value.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation

WalkThroughData

Prototype:
[ReadOnly] VARIANT WalkThroughData;

Description:

It contains the user's WalkThrough face template authentication data.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

WalkThroughLength, WalkThroughType

WalkThroughLength

Prototype:

[ReadOnly] long WalkThroughLength;

Description:

Contains the user's WalkThrough template data size.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

WalkThroughData, WalkThroughType

Walkthrough Type

Prototype:

[ReadOnly] long WalkThroughType;

Description:

Contains information about the user's WalkThrough data type. (TEMPLATE only: 13)

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

WalkThroughLength, WalkThroughType

WalkThroughData_2

Prototype:

[ReadOnly] VARIANT WalkThroughData_2;

Description:

It contains the user's WalkThrough face jpg authentication data.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

WalkThroughLength_2, WalkThroughType_2

WalkThroughLength_2

Prototype:

[ReadOnly] long WalkThroughLength_2;

Description:

Contains the user's WalkThrough jpg data size.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

WalkThroughData_2, WalkThroughType_2

WalkThroughType_2

Prototype:

[ReadOnly] long WalkThroughType_2;

Description:

Contains information about the user's WalkThrough data type. (only JPG: 12)

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

WalkThroughLength_2, WalkThroughType_2

Iris Data Length

Prototype:

[ReadOnly] long IrisDataLength;

Description:

Contains the user's iris authentication data size.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IrisData

IrisData

Prototype:

[ReadOnly] VARIANT IrisData;

Description:

It contains the user's iris authentication data.

After calling the GetUserInfoListFromTerminal/GetUserDataFromTerminal method, you can obtain it.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

Iris Data Length

MaxUserNumber

Prototype:

[ReadOnly] unsigned long MaxUserNumber;

Description:

Maximum number of users that can be registered.

It can be obtained after calling GetUserCountFromTerminal.

Related methods:

GetUserCountFromTerminal

Related properties:

N/A

RegFingerNumber

Prototype:

[ReadOnly] unsigned long RegFingerNumber;

Description:

Registered fingerprints count.

It can be obtained after calling GetUserCountFromTerminal.

Related methods:

GetUserCountFromTerminal

Related properties:

N/A

MaxFingerNumber

Prototype:

[ReadOnly] unsigned long MaxFingerNumber;

Description:

Maximum number of fingerprints that can be registered.

It can be obtained after calling GetUserCountFromTerminal.

Related methods:

GetUserCountFromTerminal

Related properties:

N/A

RegFaceNumber

Prototype:

[ReadOnly] unsigned long RegFaceNumber;

Description:

Registered face data count.

It can be obtained after calling GetUserCountFromTerminal.

Related methods:

GetUserCountFromTerminal

Related properties:

N/A

MaxFaceNumber

Prototype:

[ReadOnly] unsigned long MaxFaceNumber;

Description:

Maximum number of face data entries that can be registered.

It can be obtained after calling GetUserCountFromTerminal.

Related methods:

GetUserCountFromTerminal

Related properties:

N/A

4.3.2 Methods

This document describes the various methods of the ITerminalUserData interface.

GetUserCountFromTerminal

Prototype:

```
HRESULT GetUserCountFromTerminal(long ClientID, long TerminalID);
```

Description:

Retrieve the total number of registered users from the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related Properties

ErrorCode

Callback Event:

GetUserCount

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

AdminNumber:

Number of registered Admins.

UserNumber:

Number of registered users

GetUserDataFromTerminal

Prototype:

```
HRESULT GetUserDataFromTerminal(long ClientID, long TerminalID, long UserID);
```

Description:

Retrieves user data from the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

UserID:

User ID.

Related Properties

ErrorCode, UserID, UserName, UniqueID, AccessGroup, AccessDateType, StartAccessDate,
Start Access Time, End Access Time, End Access Date, Is Admin, Is Identified, Is And Operation,
IsFinger, IsFPCard, IsCard, IsPassword, IsCardID, Password, CardNumber, RFID,
SecurityLevel, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber,
FPSampleData, PictureDataLength, PictureData

Includes note [w7]: Added on 20250418

Callback Event:

EventGetUserData

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

DeleteUserFromTerminal

Prototype:

```
HRESULT DeleteUserFromTerminal(long ClientID, long TerminalID, long UserID);
```

Description:

Deletes user data from the specified device.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

UserID:

User ID.

Related Properties

ErrorCode

Callback Event:

DeleteUserEvent

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

UserID:

User ID.

DeleteAllUsersFromTerminal

Prototype:

```
HRESULT DeleteAllUserFromTerminal(long ClientID, long TerminalID);
```

Description:

Deletes all user data from the specified device.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related Properties

ErrorCode

Callback Event:

DeleteAllUsersEvent

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Register Face from Terminal

Prototype:

```
HRESULT RegistFaceFromTerminal(long ClientID, long TerminalID, long Opt);
```

Description:

When this command is issued, it will prompt for facial input from the designated terminal.

Until the stop command is issued, standard registration captures 5 faces and quick registration captures 3 faces.

An event occurs each time a capture is made.

Parameters:

ClientID:

The client ID of the client who requested the work..

TerminalID:

Device ID.

Opt:

Command issuance options: 0: Start registration, 1: Stop registration.

Related Properties

ErrorCode

Callback Event:

EventRegistFace

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

CurrentIndex:

Face scan index.

TotalNumber:

Maximum face scan count.

RegData:

Face scan data time.

get_FPSampleDataLength

Prototype:

HRESULT get_FPSampleDataLength(long nFingerID, long nSampleNum);

Description:

Retrieve the template data size of the finger ID specified in the last received FIR.

Parameters:

nFingerID:

The client ID of the client who requested the task.

nSampleNum:

Template number.

Each Finger ID has two sets of template data.

Related Properties

Error Code, Template Data Size

Register Iris from Terminal

Prototype:

HRESULT RegisterIrisFromTerminal(long ClientID, long TerminalID, long opt);

Description:

When this command is issued, the iris will be captured from the designated terminal.

After touching the start button on the terminal, the iris scan begins within 10 seconds.

After touching the start button on the terminal, a 10-second timeout is applied.

Even if you do not touch the start button on the terminal, it has a 60-second timeout.

The function stops when it times out or when a stop command is issued from the server.

An event occurs when scanning is complete or stopped.

Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

opt:

Command options: 0: Start registration, 1: Stop registration.

Related Properties

ErrorCode

Callback Event:

EventRegisters

Event Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

CurrentIndex:

Face scan index.

TotalNumber:

Maximum face scan count.

RegData:

Face scan data time.

Register and Walk Through Face Recognition from Terminal

Prototype:

HRESULT RegistWalkThroughFaceFromTerminal(long ClientID, long TerminalID, long opt);

Description:

When this command is issued, it will prompt for facial input from the designated terminal.

It captures one face until the stop command is issued, and an event occurs upon completion.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

opt:

Command options: 0: Start registration, 1: Stop registration.

Related Properties

ErrorCode

Callback Event:

Event Walkthrough Data

Event Parameters:

ClientID:

The client ID of the client who requested the work

TerminalID:

Device ID.

UserID:

User ID.

DataType:

jpg: 12, template: 13.

DataLength:

scan data size.

Data:

scan data.

GetUserInfoListFromTerminal

Prototype:

```
HRESULT GetUserInfoListFromTerminal(long ClientID, long TerminalID);
```

Description:

Retrieves the list of all registered user information from the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related Properties

ErrorCode

Callback Event:

EventGetUserInfoList

Event Parameters:

ClientID:

The client ID of the client who requested the work

TerminalID:

Device ID.

4.4 IAccessLogData Interface

Interface related to the function of retrieving authentication logs from terminals. This interface is used to retrieve authentication logs from terminals.

You must obtain and use the pace.

4.4.1 Properties

This section describes the various properties of the IAccessLogData interface.

CurrentIndex / TotalNumber

Prototype:

[ReadOnly]	long	currentIndex;
[ReadOnly]	long	TotalNumber;

Description:

When obtaining multiple log records, it contains the total number of records and the index of the current record.

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

GetAccessLogFromTerminal

UserID

Prototype:

[ReadOnly]	long	UserID;
------------	------	---------

Description:

It contains the user ID value.

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

GetAccessLogFromTerminal

AuthType

Prototype:
[ReadOnly] long AuthType;

Description:

It contains the authentication type value. The possible values are as follows.

0 - 1:N fingerprint authentication
1 - 1:1 fingerprint authentication
2 - Fingerprint Card Authentication
3 - Card Authentication
4 - Password Authentication

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

[GetAccessLogFromTerminal](#)

AuthMode

Prototype:
[ReadOnly] long AuthMode;

Description:

Contains the authentication Mode value. Possible values are as follows.

0-Commute
1-Leaving work
2 - General (General entry and exit)
3-Field Work
4-Return

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

[GetAccessLogFromTerminal](#)

DateTime

Prototype:

[ReadOnly] BSTR DateTime;

Description:

It contains the authentication timestamp data as a string.

The data format is "yyyy-MM-dd hh:mm:ss".

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

GetAccessLogFromTerminal

IsAuthorized

Prototype:

[ReadOnly] BOOL IsAuthorized;

Description:

The authentication result value is contained here. This value holds 0 for successful authentication and 1 for failure.

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

GetAccessLogFromTerminal

RFID

Prototype:

[ReadOnly] BSTR RFID;

Description:

When the authentication type is Card, the RFID value is stored as a string.

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

GetAccessLogFromTerminal

Picture Data Length

Prototype:

[ReadOnly] long PictureDataLength;

Description:

It contains the size value of the photo data.

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

GetAccessLogFromTerminal

Related properties:

PictureData

PictureData

Prototype:

[ReadOnly] VARIANT PictureData;

Description:

It contains photo data in binary stream format. The data length value is PictureDataLength.

It is contained within the property.

You can obtain it after calling the GetAccessLogFromTerminal method.

Related methods:

GetAccessLogFromTerminal

Related properties:

Picture Data Length

FingerImageData

Prototype:

[ReadOnly] VARIANT FingerImageData;

Description:

It contains fingerprint image data in binary stream format.

The length value of the data is contained in the FingerImageLength property.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

FingerImageLength

Prototype:

[ReadOnly] long FingerImageLength;

Description:

Contains the size data of the FingerImage.

The actual data is contained in the FingerImageData property.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

FingerImageFormat

Prototype:

[ReadOnly] BSTR Format;

Description:

It contains the data format data for FingerImage.

Only the JPG format is supported.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

Latitude

Prototype:

[ReadOnly] float Latitude;

Description:

It contains the location (latitude) information of the authentication terminal.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

Longitude

Prototype:

[ReadOnly] float Longitude;

Description:

It contains the location (longitude) information of the authentication terminal.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

IsDrinking

Prototype:

[ReadOnly] long IsDrinking;

Description:

It contains the status value indicating whether alcohol was consumed.

(Status: General access (0), Non-alcoholic (1), Alcoholic (2>

Available on terminals supporting breathalyzer functionality.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

Drinking Level

Prototype:

[ReadOnly] BSTR Drinking Level;

Description:

It contains values for blood alcohol levels.

Available on terminals supporting breathalyzer functionality.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

Thermal Burn Type

Prototype:

[ReadOnly] long Thermal Burn Type;

Description:

It contains values for the thermal imaging measurement status.

(Status: General access (0), Normal temperature (1), Abnormal temperature (2>

Available on devices supporting thermal imaging measurement functionality.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

ThermalBurn

Prototype:

[ReadOnly] BSTR ThermalBurn;

Description:

It contains values for thermal imaging metrics.

Available on devices supporting thermal imaging measurement functionality.

Available after user authentication is complete.

Related methods:

N/A

Related properties:

N/A

4.4.2 Methods

This document describes the various methods of the IAccessLogData interface.

SetPeriod

Prototype:

```
HRESULT SetPeriod(long StartYear, long StartMonth, long StartDay,  
                  long EndYear, long EndMonth, long EndDay);
```

Description:

When obtaining only authentication record information within a specified period, set the period information.

This method is called GetAccessLogCountFromTerminal and GetAccessLogFromTerminal.

Used when LogType = 3.

Parameters:

StartYear/StartMonth/StartDay:

Specify the start date.

EndYear/EndMonth/EndDay:

Specify the end date.

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

GetAccessLogCountFromTerminal

Prototype:

```
HRESULT GetAccessLogCountFromTerminal(long ClientID, long TerminalID, long LogType);
```

Description:

Retrieve the number of authentication logs for the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

LogType:

You can specify the type of log to retrieve. The possible values are as follows.

- 0 - New Log
- 1 - Logs already sent to the server
- 2 - All stored logs
- 3 - Period-based Log

When LogType = 3, first set the period information using the SetPeriod method.

Related properties

ErrorCode

Callback Event:

EventGetAccessLogCount

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

LogCount:

Number of authentication logs.

GetAccessLogFromTerminal

Prototype:

```
HRESULT GetAccessLogFromTerminal(long ClientID, long TerminalID, long LogType);
```

Description:

Retrieves the authentication logs stored on the specified terminal.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

LogType:

You can specify the type of log to retrieve. The possible values are as follows.

- 0 - New Log
- 1 - Logs already sent to the server
- 2 - All stored logs
- 3 - Period-based Log

When LogType = 3, first set the period information using the SetPeriod method.

Related Properties

ErrorCode, TotalNumber, CurrentIndex, UserID, DateTime, AuthType, AuthMode,
IsAuthorized, RFID, PictureDataLength, PictureData

Callback Event:

EventGetAccessLog

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

SetFingerImageSend

Prototype:

```
HRESULT SetFingerImageSend(bool IsSend);
```

Description:

Enable or disable the fingerprint image reception function on the terminal.

(Available only when fingerprint authentication is enabled on the server)

Parameters:

IsSend:

Fingerprint Image Reception Function Enabled/Disabled

Related Properties

N/A

Callback Event:

N/A

Event Parameters:

N/A

GetAccessLogFromTerminalEx2

Prototype:

```
HRESULT GetAccessLogFromTerminalEx2(int ClientID, int TerminalID, int LogType, int LogImage);
```

Description:

This is an extended API for the UCSAPI_GetAccessLogFromTerminal function. Retrieve log images.

A parameter has been added for.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

LogType:

Specifies the log type to retrieve. Possible values are listed in UCSAPI_GET_LOG_TYPE.

When LogType = 3, first set the period information using the SetPeriod method.

LogImage:

Specifies whether to use log images. Possible values are defined in UCSAPI_LOG_IMAGE.

Related Properties

ErrorCode, TotalNumber, CurrentIndex, UserID, DateTime, AuthType, AuthMode,

IsAuthorized, RFID, PictureDataLength, PictureData

Callback Event:

EventGetAccessLog

Event Parameters:

ClientID:

The ID of the client who requested the work.

TerminalID:

Device ID.

4.5 IAccessControlData Interface

An interface for transmitting access control data to terminals. This is used to configure access control information on terminals.

You must obtain and use the interface.

4.5.1 Properties

This section describes the various properties of the IAccessControlData interface.

4.5.2 Methods

This section describes the various methods of the IAccessControlData interface.

InitData

Prototype:

```
HRESULT InitData(void);
```

Description:

Specify whether to initialize and recreate the access control data.

Parameters:

N/A

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A.

SetTimeZone

Prototype:

```
HRESULT SetTimeZone(BSTR Code, long nIndex, long StartHour, long StartMinute,  
long EndHour, long EndMinute);
```

Description:

Add a verifiable time slot during the day.

The maximum number of time zone codes that can be added is 128, and the number of time zones that can be added to a single time zone code

The maximum is 12, and the SDK contains the additional information in an array.

Parameters:

Code:

A fixed 4-byte string representing the identifier code value for the time zone to be set.

nIndex:

Index for time zone information. This value can range from 0 to 11.

StartHour / StartMinute:

You can specify the start time of the time zone.

EndHour / EndMinute:

You can specify the end time for the time slot.

Related methods

[Set Access Control Data to Terminal](#)

Related properties

[ErrorCode](#)

Callback Event:

N/A

Event Parameters:

N/A

SetAccessTime

Prototype:

```
HRESULT SetAccessTime(BSTR Code, BSTR Sun, BSTR Mon, BSTR Tue, BSTR
Wed, BSTR Thu, BSTR Fri, BSTR Sat, BSTR Hol, BSTR Holiday);
```

Description:

Add the available access times by day of the week.

The maximum number of additional accessible time codes is 128, and the SDK stores the added information in an array.

There is.

Parameters:

Code:

A fixed 4-byte string as the identifier code value for the accessible time to be set.

Sun/Mon/Tue/Wed/Thu/Fri/Sat/Holiday:

Specify the access time slot codes for each day of the week to be used during authentication and the access time slot codes to be applied on holidays.

It can be done.

Holiday:

You can specify the holiday code set in the SetHoliday method. The specified holiday code corresponds to the Hol code.

The time zone applies.

Related methods

[Set Access Control Data to Terminal](#)

Related properties

[ErrorCode](#)

Callback Event:

N/A

Event Parameters:

N/A

SetHoliday

Prototype:

```
HRESULT SetHoliday(BSTR Code, long nIndex, long Month, long Day);
```

Description:

Add holiday information.

The maximum number of additional holiday codes is 64, and the maximum number of holidays that can be added to a single holiday code is

There are 32 of them. The SDK contains the added information in an array.

Parameters:

Code:

A fixed 4-byte string as the identifier code value for the holiday to be set.

nIndex:

The index value of the holiday to be added. This value can range from 0 to 63.

Month / Day:

You can specify the date for holidays.

Related methods

[SetAccessControlDataToTerminal](#)

Related properties

[ErrorCode](#)

Callback Event:

N/A

Event Parameters:

N/A

SetAccessGroup

Prototype:

```
HRESULT SetAccessGroup(BSTR Code, long nIndex, BSTR AccessTime);
```

Description:

Add access group information.

The maximum number of additional access group codes is 128, and the number of users that can be added to a single access group code is

The time code can have up to 4 elements. The SDK stores the additional information in an array.

Parameters:

Code:

A fixed 4-byte string as the identifier code value for the access group to be configured.

nIndex:

The index value of the access time code to be added. This value can range from 0 to 3.

AccessTime:

You can specify the access time code to be used in the access group.

Related methods

[Set Access Control Data to Terminal](#)

Related properties

[ErrorCode](#)

Callback Event:

N/A

Event Parameters:

N/A

Set Access Control Data to Terminal

Prototype:

```
HRESULT SetAccessControlDataToTerminal(long ClientID, long TerminalID, long DataType);
```

Description:

SetTimeZone, SetAccessTime, SetHoliday, SetAccessTime, SetAccessGroup for the specified terminal

Transmits access control data added by the Method to the terminal. Time period, access time, holidays, access type

Loop information must be transmitted separately. Access control information is used during authentication at the terminal. Terminal

If there is no stored access control information, no separate access control is performed.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

DataType:

You can specify the access control data to be transmitted to the terminal. The possible values are as follows.

- | | |
|-----|------------------------------|
| 0 - | Time Zone Information |
| 1 - | Holiday Information |
| 2 - | Accessible Hours Information |
| 3 - | Access Group Information |

Related methods

SetTimeZone, SetHoliday, SetAccessTime, SetAccessGroup

Related properties

ErrorCode

Callback Event:

```
HRESULT EventSetAccessControlData(long ClientID, long TerminalID, long DataType);
```

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

DataType:

Access control data type value.

SetTimeZoneToAuthProperty

Prototype:

```
HRESULT SetTimeZoneToAuthProperty(BSTR Code, BOOL Zone1, BOOL Zone2, BOOL Zone3,  
BOOL Zone4, BOOL Zone5, BOOL Zone6, BOOL Zone7, BOOL Zone8, BOOL Zone9,  
BOOL Zone10, BOOL Zone11, BOOL Zone12, BOOL Reserved1, BOOL Reserved2,  
BOOL Reserved3, BOOL Reserved4);
```

Description:

Enable/disable the use of the authenticatable time zone for the specified TimeZone.

Parameters:

Code:

A fixed-size string with the identifier code value for the time zone, UCSAPI_DATA_SIZE_CODE4(4).

Zone1 ~ Zone12:

Usage flag value for each time zone.

Related methods

N/A

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetAuthProperty

Prototype:

```
HRESULT SetAuthProperty(BSTR Code, BOOL AndOperation, BOOL Finger, BOOL TOC,  
BOOL Password, BOOL Card, BOOL Face, BOOL Reserved1, BOOL Reserved2);
```

Description:

Set the allowed authentication type for the specified TimeZone.

Parameters:

Code:

A fixed-size string with the identifier code value for the time zone, UCSAPI_DATA_SIZE_CODE4(4).

AndOperation:

You can specify that each authentication type be used in combination with others via AND or OR.

Here, set 1 for an AND combination and 0 for an OR combination.

For details, refer to the user attributes in Chapter 1.5, Terminology.

Finger:

Fingerprint authentication.

Table of Contents:

Password:

Password authentication.

Card:

Card authentication.

Face:

Face authentication.

Related methods

N/A

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

4.6 IServerAuthentication Interface

An interface for performing server authentication. The application responds to user authentication requests from the terminal.

To respond, you must obtain and use this interface.

4.6.1 Properties

This section describes the various properties of the IServerAuthentication interface.

DeviceID

Prototype:

[ReadOnly] long DeviceID;

Description:

The input device ID value of the device requesting authentication is contained here.

After calling an event related to server authentication, it can be obtained.

Related Events:

Event Authentication Type with User ID

Event Authentication Type with Unique ID

Event Verification Card

EventVerifyFinger1to1

EventVerifyFinger1toN

Event Verification Password

4.6.2 Methods

This document describes the various methods of the IServerAuthentication interface.

SetAuthenticationType

Prototype:

```
HRESULT SetAuthType(BOOL AndOperation, BOOL Finger, BOOL FPCard, BOOL Password,  
BOOL Card, BOOL CardID);
```

Description:

You can specify the user's authentication type. Each authentication type is determined by the AndOperation Flag.

It can be used in combination with AND or OR. Must be used before calling the SendAuthInfoToTerminal method.
shall be done.

Parameters:

AndOperation:

If this value is True, each authentication type can be authenticated using an AND combination. If False,
Authentication is possible using OR combinations.

Finger:

Fingerprint authentication can be enabled.

FPCard:

Fingerprint card authentication can be enabled. Fingerprint cards store fingerprint information on smart cards.

This is a method of certification.

Password:

Password authentication can be enabled.

Card:

Card authentication can be enabled.

CardID:

You can designate the card number as an ID. Use RFID as a UserID or UniqueID. can be designated. CardID does not use the card's RFID as an authentication method, but only the UserID. This refers to using it as an identifier. It must be specified in combination with other authentication types using an AND operator. You must.

Related methods

[SendAuthenticationInformationToTerminal](#)

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SendAuthenticationInformationToTerminal

Prototype:

```
HRESULT SendAuthInfoToTerminal(long TerminalID, long UserID, BOOL  
IsAccessibility, long ErrorCode);
```

Description:

terminal Server CertificationMode Dongjak District application The program is EventAuthenticationTypeWithUserID,

EventAuthTypeWithUniqueId Immediately transmits the user's authentication information to the terminal in response to the event.

You must send it. You must use it after calling the SetAuthType method.

Parameters:

TerminalID:

Device ID.

UserID:

The ID of the user who attempted authentication.

IsAccessibility:

Specifies a flag value indicating whether the user is authorized to enter. If this value is False, the terminal will perform authentication.

Process the ¶.

Related Properties

ErrorCode

Callback Event:

N/A

Event Parameters

N/A

SendAuthResultToTerminal

Prototype:

```
HRESULT SendAuthResultToTerminal(long TerminalID, long UserID, BOOL  
IsAccessibility, BOOL IsVisitor, BOOL IsAuthorized, BSTR AuthorizedTime, long ErrorCode);
```

Description:

The terminal notifies the application of the following events for user authentication.
EventVerifyCard, EventVerifyPassword, EventVerifyFinger_1_TO_1, EventVerifyFinger_1_TO_N at this time
The application must immediately transmit the user's authentication result to the terminal.

Parameters:

TerminalID:

Device ID.

UserID:

The ID of a user who has been authenticated or attempted to authenticate.

IsAccessibility:

Sets a flag indicating whether the user is authorized to enter. If this value is False, the terminal fails authentication.
Process.

IsVisitor:

Specifies whether it is a visitor.

IsAuthorized:

Specifies whether authentication was successful.

Authorized Time:

Specify the authentication time. This value must be a string in the format "yyyy-MM-dd hh:mm:ss".

ErrorCode:

You can specify the error code that occurs during authentication. If this value is 0, there is no error. For details,
Refer to the ErrorCode table.

Related Properties

ErrorCode

Callback Event:

EventVerifyCard, EventVerifyPassword, EventVerifyFinger_1_TO_1, EventVerifyFinger_1_TO_N

Event Parameters

N/A

SendAntipassbackResultToTerminal

Prototype:

```
HRESULT SendAntipassbackResultToTerminal(long TerminalID, long UserID, BOOL  
bResult);
```

Description:

When the terminal has the Antipassback option enabled and performs authentication on the terminal, the user's application has notified the EventAntipassback event to obtain the current Antipassback status. At this point, the application immediately transmits the user's antipassback status to the terminal, stored in bResult. It must be done. This event occurs only when the terminal performs authentication on the terminal.

Parameters:

TerminalID:

Device ID.

UserID:

The ID of a user who has been authenticated or attempted to authenticate.

bResult:

Access permission status based on antipassback condition. If access is permitted, it holds a value of 1.

Related Properties

ErrorCode

Callback Event:

EventAntipassback

Event Parameters

N/A

SetAuthenticationTypeEx

Prototype:

```
HRESULT SetAuthTypeEx(BOOL Face, BOOL MobileKey, BOOL Reserved1, BOOL Iris,  
BOOL Reserved3, BOOL Reserved4, BOOL Reserved5, BOOL Reserved6);
```

Description:

This is an extended API for the SetAuthType function.

Face and MobileKey, along with iris authentication, have been added as additional supported types.

Parameters:

Face:

Face authentication can be enabled.

MobileKey:

MobileKey authentication can be enabled.

Iris:

Iris authentication can be enabled.

Related methods

[SendAuthenticationInformationToTerminal](#)

Related properties

N/A

Callback Event:

N/A

Event Parameters:

N/A

4.7 ITerminalOption Interface

terminal

Interface for option settings.

Applications must obtain and use this interface to set or retrieve terminal options.

The terminal options consist of default options, network settings, lock settings, and holiday settings.

4.7.1 Properties

Describes the various properties of the ITerminalOption interface.

flagSecuLevel / flagInputIDLength / flagAutoEnterKey / flagSound / flagAuthentication /
flagApplication / flagAntipassback / flagNetwork / flagInputIDType / flagAccessLevel / flagPrintText
/ flagSchedule

Prototype:

[WriteOnly] long flagSecuLevel-flagSchedule;

Description:

It holds the reference flag value for the option item to be set. When this flag value is 1,

The terminal can reference the values of all items. However, it can only set values for items whose flag is set to 1.

Use.

SetOptionToTerminal Method Before calling the method, set the corresponding flag and its value.

Related methods:

SetOptionToTerminal

SecurityLevel_1To1 / SecurityLevel_1ToN

Prototype:

[Read/Write] long SecurityLevel_1To1;
[Read/Write] long SecurityLevel_1ToN;

Description:

It holds the 1:1 and 1:N authentication level values to be used during terminal authentication. To set this value, use flagSecuLevel.

The value must be set to 1. This value can take the following values.

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

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, you can reference it.

Related methods:

[SetOptionToTerminal](#)
[GetOptionFromTerminal](#)

Input ID Length

Prototype:

[Read/Write] long Input ID Length;

Description:

It holds the length value of the ID entered on the terminal. When using UserID, a maximum of 8 can be specified.

When using UniqueID, you can specify a maximum value of 20. To set this value, use flagInputIDLength.

The value must be set to 1.

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

[SetOptionToTerminal](#)
[GetOptionFromTerminal](#)

AutoEnterKey

Prototype:

[ReadWrite] long AutoEnterKey;

Description:

Holds the value indicating whether the terminal's auto-enter key is enabled. This function inputs keys for the length specified by InputIDLength.

This feature automatically inputs the Enter key when it is present. To set this value, use flagAutoEnterKey.

The value must be set to 1.

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

[SetOptionToTerminal](#)

[GetOptionFromTerminal](#)

Sound

Prototype:

[ReadWrite] long Sound;

Description:

Holds the sound volume value of the terminal. The volume value can be set from 0 to 20. The terminal's sound

To set it to mute, specify 0. To set this value, set the flagSound value to 1.

You must

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

[SetOptionToTerminal](#)

[GetOptionFromTerminal](#)

Authentication

Prototype:

[Read/Write] long Authentication;

Description:

Holds the terminal authentication method value. Possible values are as described in Section 1.5, "Terminal Authentication Method."

Refer to "flagAuthentication". To set this value, the flagAuthentication value must be set to 1.

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

SetOptionToTerminal

GetOptionFromTerminal

Application

Prototype:

[Read/Write] long Application;

Description:

The terminal program has a mode value. Terminals are used for functions such as access control, attendance tracking, and meal management.

It can be used as. For possible values, refer to "Terminal Program Mode" in Chapter 1.6, Terminology.

To set this value, the flagApplication value must be set to 1.

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

SetOptionToTerminal

GetOptionFromTerminal

Antipassback

Prototype:

[Read/Write] long Antipassback;

Description:

Possesses the anti-passback level value of the terminal

To set this value, the flagAntipassback value must be set to 1.

This value can have the following values.

0- Not used

1- Allow access when the connection to the server is lost

2- Access denied when connection to the server is lost

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

[SetOptionToTerminal](#)

[GetOptionFromTerminal](#)

[NetworkType](#) / [TerminalIP](#) / [Subnet](#) / [Gateway](#) / [ServerIP](#) / [Port](#)

Prototype:

[ReadOnly]	long	NetworkType;
[ReadOnly]	BSTR	TerminalIP;
[ReadOnly]	BSTR	Subnet;
[ReadOnly]	BSTR	Gateway;
[ReadOnly]	BSTR	ServerIP;
[ReadOnly]	long	Port;

Description:

Contains the device's network-related values. These values cannot be set by the application.

After calling the GetOptionFromTerminal method, it can be referenced.

[NetworkType](#)

It has a type value for the IP address. If the value is 0, it supports a static IP; if it is 1, it supports a dynamic IP.

[Terminal IP](#)

It has a terminal IP address.

[Subnet](#)

It has the Subnet Mask value of the terminal.

Gateway

It has the Gateway value of the terminal.

ServerIP

The terminal holds the IP address of the server to connect to.

Port

The terminal has a server port value to connect to. This value uses the default value of 9870.

Related methods:

[SetOptionToTerminal](#)

Input ID Type

Prototype:

[Read/Write] long InputIDType;

Description:

It holds the type value of the ID entered during authentication on the terminal. The possible values are as follows.

To set this value, the flagInputIDType value must be set to 1.

This value can have the following values.

0- UserID

1- UniqueID

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

[SetOptionToTerminal](#)

[GetOptionFromTerminal](#)

AccessLevel

Prototype:

[Read/Write] long AccessLevel;

Description:

It has an access level value. This function restricts the authentication type entered on the terminal to the specified type.

Only allows authentication. Possible values are as follows. The default value is 0.

To set this value, the flagAccessLevel value must be set to 1.

This value can take the following values.

0 - No limit

1- Only fingerprint and password authentication are permitted.

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, it can be referenced.

Related methods:

[SetOptionToTerminal](#)

[GetOptionFromTerminal](#)

PrintText

Prototype:

[Read/Write] long PrintText;

Description:

This is a buffer array containing the string to be printed on the water printer connected to the terminal.

This value can be used when a water printer is connected to the terminal.

To set this value, the flagPrintText value must be set to 1.

This value must be set before calling the SetOptionToTerminal method, either by setting the corresponding flag and value or

After calling the GetOptionFromTerminal method, you can reference it.

Related methods:

[SetOptionToTerminal](#)

[GetOptionFromTerminal](#)

`IsUse / StartHour / StartMinute / EndHour / EndMinute`

Prototype:

<code>[ReadOnly]</code>	<code>long</code>	<code>IsUse;</code>
<code>[ReadOnly]</code>	<code>long</code>	<code>StartHour;</code>
<code>[ReadOnly]</code>	<code>long</code>	<code>StartMinute;</code>
<code>[ReadOnly]</code>	<code>long</code>	<code>EndHour;</code>
<code>[ReadOnly]</code>	<code>long</code>	<code>EndMinute;</code>

Description:

It holds the Time Zone value for the Lock or Open schedule of the terminal.

This value can be referenced after calling the `GetDaySchedule` method.

IsUse

Specifies whether the time zone is valid. The time zone is valid only when this value is 1.

StartHour

It holds the start time value of the time zone. This value can range from 0 to 23.

StartMinute

It holds the starting minute value of the time zone. This value can range from 0 to 59.

EndHour

It holds the end time value of the time zone. This value can range from 0 to 23.

EndMinute

It holds the ending minute value of the time zone. This value can range from 0 to 59.

Related methods:

`GetOptionFromTerminal`

`GetDaySchedule`

Month / Day

Prototype:

[ReadOnly]	long	Month;
[ReadOnly]	long	Day;

Description:

It holds the holiday information value of the terminal.

This value can be referenced after calling the GetHoliday method.

Month

It holds the Month value from the date information. This value can range from 1 to 12.

Day

It holds the Day value from the date information. This value can range from 1 to 31.

Related methods:

[GetOptionFromTerminal](#)
[GetHoliday](#)

HolidayType

Prototype:

[ReadOnly]	long	HolidayType;
------------	------	--------------

Description:

The terminal has a holiday type value. This value can be between 1 and 12.

This value can be referenced after calling the GetHoliday method.

Related methods:

[GetOptionFromTerminal](#)
[GetHoliday](#)

Port

Prototype:

[Read/Write] long Port;

Description:

It holds the terminal's network port settings.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

ServerIP

Prototype:

[Read/Write] BSTR ServerIP;

Description:

It holds the terminal's network IP settings.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

Network Type

Prototype:

[Read/Write] long NetworkType;

Description:

It holds the network type setting value of the terminal.

It holds the type value of the IP address. If the value is 0, it supports a static IP; if it is 1, it supports a dynamic IP.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagSchedule

Prototype:
[Read/Write] bool Schedule;

Description:

To set an option item on the terminal, set the flag value for that item to True and set the item value to

Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagPrintText

Prototype:
[Read/Write] bool Schedule;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to

Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagAccessLevel

Prototype:
[Read/Write] bool AccessLevel;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to

Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagInputIDType

Prototype:
[ReadWrite] bool InputIDType;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagServer

Prototype:
[ReadWrite] bool Server;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagNetwork

Prototype:
[ReadWrite] bool Network;

Description:

To set an option item on the terminal, set the flag value for that item to True and set the item value to Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagAntipassback

Prototype:

[Read/Write] bool Antipassback;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagApplication

Prototype:

[Read/Write] bool Application;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagAuthentication

Prototype:

[Read/Write] bool Authentication;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagSound

Prototype:

[ReadWrite] bool Sound;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to

Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagAutoEnterKey

Prototype:

[ReadWrite] bool AutoEnterKey;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to

Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagInputIDLength

Prototype:

[ReadWrite] bool Input ID Length;

Description:

To set an option item on the terminal, set the flag value for that item to True and set the item value to

Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

flagSecuLevel

Prototype:

[Read/Write] bool SecuLevel;

Description:

To set an option item on the terminal, set the flag value of that item to True and set the item value to

Designates.

For a description of this item, refer to the description of the UCSAPI_TERMINAL_OPTION structure.

Related methods:

GetOptionFromTerminal, SetOptionToTerminal

4.7.2 Methods

This document describes the various methods of the ITerminalOption interface.

SetOptionToTerminal

Prototype:

```
HRESULT SetOptionToTerminal(LONG ClientID, LONG TerminalID);
```

Description:

Configure the terminal's options. Set the relevant Properties and Methods you wish to configure before calling the Method.

You must fill in the corresponding value using it.

Refer to Chapter 4.7.1 for the relevant Properties and Methods.

Parameters:

ClientID:

The client ID of the client who requested the task.

Terminal ID:

Device ID.

Related methods

SetHoliday

Set Daily Schedule

Related properties

flagSecuLevel ~ flagSchedule

SecurityLevel

Input ID Length

AutoEnterKey

Sound

Authentication

Application

Antipassback

Input ID Type

AccessLevel

PrintText

NetworkType

Terminal IP

Subnet

Gateway

ServerIP

Port

Callback Event:

Event Set Terminal Option

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

GetOptionFromTerminal

Prototype:

HRESULT GetOptionFromTerminal(LONG ClientID, LONG TerminalID);

Description:

Reads option values from the terminal. After calling the Method, the EventGetTerminalOption event

You can check the corresponding values by referring to Properties and Method.

Refer to Chapter 4.7.1 for the relevant Properties and Methods.

Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Related methods

GetHoliday

GetDaySchedule

Related properties

SecurityLevel_1To1

SecurityLevel_1ToN

Input ID Length

AutoEnterKey

Sound

Authentication

Application

Antipassback

Input ID Type

AccessLevel

PrintText

NetworkType

Terminal IP

Subnet

Gateway

ServerIP

Port

Callback Event:

GetTerminalOption

Event Parameters:

ClientID:

The client ID of the client who requested the task.

TerminalID:

Device ID.

Set Daily Schedule

Prototype:

```
HRESULT SetDaySchedule(BOOL Initialize, LONG DayOfWeek, LONG ScheduleType, LONG  
Index, LONG StartHour, LONG StartMinute, LONG EndHour, LONG EndMinute);
```

Description:

This method sets the Lock or Open time zone to the terminal before calling the SetOptionToTerminal method.

Used for adding.

Parameters:

Initialize:

Specify whether to reset the Lock & Open Schedule and create a new one.

If this value is False (=0), the Time zone data being added now is internally generated day

It continues to add to the site, creating multiple time zone data entries.

Setting this value to True (=1) will clear all existing time zones and create new ones.

A schedule can have three Lock Time zones and three Open Time zones per day of the week.

DayOfWeek:

Specify the day of the week and holiday type.

0 – Sun

1 – Mon

2 – Tue

3 – Wed

4 – Thu

5 – Fri

6 – Sat

7 – Holiday 1

8 – Holiday 2

9 – Holiday 3

ScheduleType:

Specifies whether to lock or unlock.

0 – Lock

1 – Open

Index:

It has the Index value for the Lock and Open Time zone.

1 – Lock 1 or Open 1

2 – Lock 2 or Open 2

3 – Lock 3 or Open 3

StartHour

It holds the start time value of the time zone. This value can range from 0 to 23.

StartMinute

It holds the starting minute value of the time zone. This value can range from 0 to 59.

EndHour

It holds the end time value of the time zone. This value can range from 0 to 23.

EndMinute

It holds the ending minute value of the time zone. This value can range from 0 to 59.

Related methods

[SetOptionToTerminal](#)

Related properties

N/A

Callback Event:

N/A

Event Parameters:

N/A

GetDaySchedule

Prototype:

```
HRESULT GetDaySchedule(LONG DayOfWeek, LONG ScheduleType, LONG Index);
```

Description:

This method is called after the GetOptionFromTerminal method is invoked within EventGetTerminalOption.

Robu

Calls to check the Schedule information read from the terminal.

After calling the method, reference the IsUse, StartHour, StartMinute, EndHour, and EndMinute properties.

Parameters:

DayOfWeek:

Specify the day of the week and holiday type.

- 0 – Sun
- 1 – Mon
- 2 – Tue
- 3 – Wed
- 4 – Thu
- 5 – Fri
- 6 – Sat
- 7 – Holiday 1
- 8 – Holiday 2
- 9 – Holiday 3

ScheduleType:

Specifies whether to lock or unlock.

- 0 – Lock
- 1 – Open

Index:

It has the Index value for the Lock and Open Time zone.

- 1 – Lock 1 or Open 1
- 2 – Lock 2 or Open 2

3 – Lock 3 or Open 3

Related methods

GetOptionFromTerminal

Related properties

IsUse, StartHour, StartMinute, EndHour, EndMinute

Callback Event:

N/A

Event Parameters:

N/A

SetHoliday

Prototype:

HRESULT SetHoliday(BOOL Initialize, LONG HolidayType, LONG Month, LONG Day);

Description:

This method was used to add holidays before calling the SetOptionToTerminal method.

Holidays can have three types, and a time zone can be specified for each type.

Holidays can be added up to a maximum of 100.

Parameters:

Initialize:

Specify whether to reset the holiday and create a new one.

If this value is False (=0), the Holiday data being added now is an internally generated day.

It continues to add to the field, creating multiple Holiday data entries.

Setting this value to True (=1) will delete all existing Holidays and create new ones.

HolidayType:

Specify the Holiday Type as 1, 2, or 3.

1 – Holiday 1

2 – Holiday 2

3 – Holiday 3

Month

It holds the Month value from the date information. This value can range from 1 to 12.

Day

It holds the Day value from the date information. This value can range from 1 to 31.

Related methods

SetOptionToTerminal

Related properties

N/A

Callback Event:

N/A

Event Parameters:

N/A

GetHoliday

Prototype:

HRESULT GetHoliday(LONG Index);

Description:

This method is called after the GetOptionFromTerminal method is invoked within EventGetTerminalOption.

Calls to verify the Holiday information read from the terminal.

After calling the method, reference the HolidayType, Month, and Day properties.

Since up to 100 holidays can be added, you can iterate by changing the index from 0 to 99.

You must call the method.

Parameters:

[Index:](#)

Specify the index value for the Holiday to retrieve. This value can range from 0 to 99.

Related methods

[GetOptionFromTerminal](#)

Related properties

[Holiday Type](#), [Month](#), [Day](#)

Callback Event:

N/A

Event Parameters:

N/A

[Clear](#)

Prototype:

`Clear HRESULT();`

Description:

This method clears all internal option setting data.

Before calling [GetOptionFromTerminal](#) or after calling [SetOptionToTerminal](#), internal configuration data

Use this when you want to initialize.

Parameters:

Related methods

[GetOptionFromTerminal](#)

[SetOptionToTerminal](#)

Related properties

Callback Event:

N/A

Event Parameters:

N/A

get_ACUStatusValue

Prototype:

```
long get_ACUStatusValue(long StatusIndex, long ValueIndex)
```

Description:

This property acquires the ACU Status value when the EventACUStatus event is received.

(See UCSAPI_ACU_STATUS_INFO)

StatusIndex

Set the type of status you wish to acquire.

#define UCSAPI_ACU_STATUS_PARTITION	0
#define UCSAPI_ACU_STATUS_ZONE	1
#define UCSAPI_ACU_STATUS_LOCK	2
#define UCSAPI_ACU_STATUS_READER	3

ValueIndex

Set the index of the value to be obtained for each StatusIndex. (Base value is 0)

The maximum value for the setting is fixed, so be careful not to exceed it when entering.

#define MAX_ACU_PARTITION	4
#define MAX_ACU_ZONE	8
#define MAX_ACU_LOCK	4
#define MAX_ACU_READER	8

ACUGetReaderVersion

Prototype:

```
HRESULT ACUGetReaderVersion(long Index, long *pID, long *pType, long *pHW, long *pMajor,
long *pMinor, long *pCustom1, long *pCustom2, long *pOrder);
```

Description:

This method acquires the ReaderVersion value from the ACU Status when receiving the EventACUStatus event.

(See UCSAPI_ACU_STATUS_INFO)

Index:

Set the index value of the Reader you wish to acquire.

The maximum value for the setting is fixed, so be careful not to exceed it when entering.

Reference Parameters:

It is a reference-type variable that acquires values for each item.

GetOptionFromACU

Prototype:

HRESULT GetOptionFromACU(LONG ClientID, LONG TerminalID);

Description:

Reads the option value from the ACU. After the method call, in the EventGetOptionFromACU event.

You can check the Option value by referring to Properties and Method.

(See UCSAPI_ACU_OPTION)

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Related properties

ACUNetType, ACUNetSPort

ACU Reader Type

ACU Reader Open Time

ACU Reader Mode

ACUUDPLockFlag

ACUUDPSitekey

ACUUDPPassword

Related methods

ACU Get Network Address

ACU Get Part Lock

ACU Get Reader Passback

ACU Get Partition

ACU Get Zone

ACU Get Program Option

ACU Get Door Option

ACU Get Input Option

ACU Get System Option

Callback Event:

GetOptionFromACU

SetOptionToACU

Prototype:

HRESULT SetOptionToACU(LONG ClientID, LONG TerminalID);

Description:

Set the ACU options. The Properties and Methods you wish to configure before the Method call.

You must fill in the corresponding value using it.

(See UCSAPI_ACU_OPTION)

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Related properties

ACU Reader Type

ACU Reader Open Time

ACU Reader Mode

ACUUDPLockFlag

ACUUDPSitekey

ACUUDPPassword

Related methods

Clear ACU Option Flag

Clear ACU Option Data

SetACUOptionFlag

ACUSetNetAddress

ACUSetPartLock

ACUSetReaderPassback

ACUSetPartition

ACUSetZone

ACUSetProgramOption

ACUSetDoorOption

ACUSetInputOption

ACUSetSystemOption

Callback Event:

Set Event Set Option to ACU

GetLockScheduleFromACU

Prototype:

```
HRESULT GetLockScheduleFromACU(LONG ClientID, LONG TerminalID, LONG LockIndex);
```

Description:

Reads the Schedule value for the Lock designated by the ACU.

After calling the method, in the EventGetLockScheduleFromACU event, the Properties and Method

You can check the Lock Schedule value by referring to it.

For schedule-related matters, please refer to GetDaySchedule.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Related properties

Please refer to GetDaySchedule related to TerminalOption.

Related methods

Please refer to GetDaySchedule related to TerminalOption.

Callback Event:

GetLockScheduleFromACU

SetLockScheduleToACU

Prototype:

```
HRESULT SetLockScheduleToACU(LONG ClientID, LONG TerminalID, long LockIndex);
```

Description:

Set the schedule for the lock designated in the ACU.

Before calling the method, the Schedule value for the lock is set using its Properties and Method.

This value must be filled in. For schedule-related matters, refer to SetDaySchedule.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Related properties

Please refer to SetDaySchedule related to TerminalOption.

Related methods

Please refer to SetDaySchedule related to TerminalOption.

Callback Event:

EventSetLockScheduleFromACU

ClearSirenConfig

Prototype:

HRESULT ClearSirenConfig();

Description:

Initializes the Siren Config value and Siren Count value

Callback Event:

N/A

SetSirenConfig

Prototype:

HRESULT SetSirenConfig(BYTE Hour, BYTE Minute, BYTE Duration,
BYTE Sun, BYTE Mon, BYTE Tue, BYTE Wed, BYTE Thu, BYTE Fri, BYTE Sat, BYTE OffHoliday);

Description:

Set the Siren Config value. This internally increases the Siren Count value.

Callback Event:

N/A

SetSirenToTerminal

Prototype:

HRESULT SetSirenToTerminal(LONG ClientID, LONG TerminalID);

Description:

Transmit the Siren Config value set by SetSirenConfig to the terminal.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Related methods

Clear Siren Configuration Set Siren Configuration

Callback Event:

Set Event Siren to Terminal

GetSirenFromTerminal

Prototype:

```
HRESULT GetSirenFromTerminal(LONG ClientID, LONG TerminalID);
```

Description:

Retrieve the Siren Config value set on the designated terminal.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Related methods

GetSirenConfig

Callback Event:

Get Siren from Terminal

GetSirenConfig

Prototype:

```
HRESULT GetSirenConfig(BYTE index, BYTE* Hour, BYTE* Minute, BYTE* Duration,
    BYTE* Sun, BYTE* Mon, BYTE* Tue, BYTE* Wed,
    BYTE* Thu, BYTE* Fri, BYTE* Sat, BYTE* OffHoliday);
```

Description:

When the terminal's settings are obtained, the EventGetSirenFromTerminal event is triggered.

At this point, the Siren Config value of the terminal is obtained through this method.

The index value increases starting from 0 by the number of Siren Counts issued by EventGetSirenFromTerminal.

Specify to obtain the Siren Config value.

Callback Event:

N/A

4.8 ISmartCardLayout Interface

An interface for setting the card layout to be applied when reading smart cards on the terminal.

4.8.1 Properties

This section describes the various properties of the ISmartCardLayout interface.

SectorNumber

Prototype:

[ReadOnly] long SectorNumber

Description:

Retrieve the number of SectorLayouts currently set in the COM.

Each time SetSectorLayout is called, it increments by 1, and when ClearSectorLayout is called, it resets to 0.

Related methods:

Clear Sector Layout

SetSectorLayout

CardType

Prototype:

[ReadOnly] long CardType

Description:

Specifies the card type. See UCSAPI_SMARTCARD_TYPE.

Related methods:

SetSmartCardLayoutToTerminal

ReadType

Prototype:

[ReadOnly] long ReadType

Description:

Specifies the card type. See UCSAPI_SMARTCARD_READTYPE.

Related methods:

[SetSmartCardLayoutToTerminal](#)

SerialFormat

Prototype:

[ReadOnly] long SerialFormat

Description:

Specifies the card type. See UCSAPI_SMARTCARD_SERIALFORMAT.

Related methods:

[SetSmartCardLayoutToTerminal](#)

4.8.2 Methods

This document describes the various methods of the ISmartCardLayout interface.

Clear Sector Layout

Prototype:

HRESULT ClearSectorLayout();

Description:

Initialize the SectorLayout Data currently stored in the COM.

Parameters:

N/A

Related methods

SetSectorLayout

SetSmartCardLayoutToTerminal

Related properties

SectorNumber

Callback Event:

N/A

SetSectorLayout

Prototype:

```
HRESULT SetSectorLayout(LONG Sector, LONG KeyType, BSTR KeyData,  
    LONG Block, LONG StartPoint, LONG DataLength, BYTE Aid0, BYTE Aid1)
```

Description:

Set the sector layout to be applied to the smart card layout in COM.

Each time it is set, the SectorNumber inside the COM increases by one.

Parameters:

Sector:

Applicable Sector Number. The maximum value is 127 (for an 8K-capacity smart card).

KeyType:

Specify the type of key data required to access the designated sector.

#define UCSAPI_SMARTCARD_KEYTYPE_A	0x60
#define UCSAPI_SMARTCARD_KEYTYPE_B	0x61

KeyData:

Specify the key data to access the relevant sector.

The specified value is in Hex String format (e.g., "000000FFFFFF");

Block:

This is the block number containing actual data. (Range: 0~3)

StartPoint:

Specifies the starting position within the block where the actual data is located.

DataLength:

Specifies the actual length of the data to be applied.

Aid0, Aid1:

The AID value applied when the read type is UCSAPI_SMARTCARD_READTYPE_MAD.

Related methods

Clear Sector Layout

SetSmartCardLayoutToTerminal

Related properties

SectorNumber

Callback Event:

N/A

SetSmartCardLayoutToTerminal

Prototype:

```
HRESULT SetSmartCardLayoutToTerminal(LONG ClientID, LONG TerminalID,  
                                     LONG CardType, LONG ReadType, LONG SerialFormat);
```

Description:

Transmit the sector information of the COM saved with the given parameter and SetSectorLayout to the terminal.

Set the card layout to be applied when reading cards on the terminal.

Parameters:

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

CardType:

Specify the card type. Refer to the following Define.

```
typedef UCSAPI_UINT32 UCSAPI_SMARTCARD_TYPE;
```

#define UCSAPI_SMARTCARD_TYPE_DATA	0
#define UCSAPI_SMARTCARD_TYPE_FINGER	1

ReadType:

Specify the type of data read by the card. Refer to the following Define.

```
typedef UCSAPI_UINT32 UCSAPI_SMARTCARD_READTYPE;
```

```
#define UCSAPI_SMARTCARD_READTYPE_SERIAL          0
#define UCSAPI_SMARTCARD_READTYPE_DATA             1
#define UCSAPI_SMARTCARD_READTYPE_MAD              2
```

SerialFormat:

When reading the serial number, specify the format in which the serial number is displayed. Refer to the following Define.

typedef UCSAPI_UINT8 UCSAPI_SMARTCARD_SERIALFORMAT;

```
#define UCSAPI_SMARTCARD_SERIALFORMAT_DEFAULT      0
#define UCSAPI_SMARTCARD_SERIALFORMAT_HEXA         1
#define UCSAPI_SMARTCARD_SERIALFORMAT_DECIMAL       2
#define UCSAPI_SMARTCARD_SERIALFORMAT_35DECIMAL     3
```

Related properties

[SetSectorLayout](#)

Related methods

Callback Event:

[EventSetSmartCardLayout](#)

4.9 Events of COM

This document describes the various events generated by the COM module.

Event User File Upgrading

Prototype:

```
HRESULT EventUserFileUpgrading(
    long ClientID,
    long TerminalID,
    long CurrentIndex,
    long TotalNumber);
```

Description:

This event notifies the application of the progress of a user file download.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

CurrentIndex:

It contains the index value of the data block currently being transmitted.

TotalNumber:

It contains the total number of data blocks to be transmitted.

Reference

SendUserFileToTerminal

EventUserFileUpgraded

Prototype:

HRESULT EventUserFileUpgraded(

```
    long ClientID,  
    long TerminalID);
```

Description:

This event notifies the application that the user file download has completed.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Reference

SendUserFileToTerminal

EventRegistFace

Prototype:

HRESULT EventRegistFace (

```
    long ClientID,  
    long TerminalID,  
    long CurrentIndex,  
    long TotalNumber,  
    VARIANT RegFaceData);
```

Description:

This event is used to notify the progress of face registration via the terminal.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

CurrentIndex:

Contains the index value of the currently registered face. The starting value is 1, and it holds a value of 0 when canceled.

TotalNumber:

Contains the total number of faces that need to be registered. Holds a value of 0 when canceled.

For standard registration, there are five items, and for simplified registration, there are three items.

RegFaceData:

This is facial data containing feature information for the input facial image. The data format is a byte array.

Reference

Register Face from Terminal

Event ACU Status

Prototype:

```
HRESULT EventACUStatus (
    long ClientID,
    long TerminalID,
    Long Notice,
    VARIANT binStatus,
    BSTR strStatus);
```

Description:

This is an event that periodically notifies you of the ACU's status.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Notice:

It indicates that there is a problem with the current state of the ACU.

binStatus:

This is a value representing a binary array listing the ACU_STATUS structure containing ACU status values.

strStatus:

This is a string representing the binStatus array converted to a HexString. (Its length is twice that of binStatus.)

Reference

SetDoorStatusToACU

GetEventLockScheduleFromACU

Prototype:

HRESULT EventGetLockScheduleFromACU(long ClientID, long TerminalID, long LockIndex);

Description:

The SDK module triggers this event in response to the application's ACU Lock Schedule request.

Notify via the application.

When this event occurs, the Schedule value can be obtained through the relevant property and method.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

LockIndex:

This is the index value for the lock whose schedule you wish to obtain.

Reference

GetLockScheduleFromACU

EventSetLockScheduleToACU

Prototype:

HRESULT EventSetLockScheduleToACU(long ClientID, long TerminalID);

Description:

The SDK module triggers this event as a result of the application's ACU Lock Schedule setting.

Notify via the application.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Reference

SetLockScheduleToACU

Set Event Siren to Terminal

Prototype:

HRESULT EventSetSirenToTerminal(long ClientID, long TerminalID);

Description:

The SDK module notifies this event as a result of setting the Siren Config value on the terminal.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Reference

ClearSirenConfig

SetSirenConfig

SetSirenToTerminal

Get Siren from Terminal

Prototype:

HRESULT EventGetSirenFromTerminal(long ClientID, long TerminalID);

Description:

When the SDK module obtains the Siren Config value from the terminal, it notifies this event as a result.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Reference

GetSirenFromTerminal

GetSirenConfig

EventSetSmartCardLayout

Prototype:

HRESULT EventSetSmartCardLayout(long ClientID, long TerminalID);

Description:

The SDK module notifies this event as a result of the smart card layout setting from the terminal.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

Reference

SetSmartCardLayoutToTerminal

GetEventTerminalTime

Prototype:

HRESULT EventGetTerminalTime(long TerminalID);

Description:

COM notifies the API of this event when a terminal requests the current time. The API handles this event.

If notified, you can set the current time using the SetTerminalTime method if necessary.

If no settings are specified in the API, COM transmits the system time to the terminal.

Event Parameters

TerminalID: 1.6 See Terminology Explanation

Reference

SetTerminalTime

GetMinutiaeFromTerminal

Prototype:

HRESULT EventGetFpMinutiaeFromTerminal (LONG ClientID, LONG TerminalID, BYTE minType,
BYTE minCount, BYTE matching, LONG minSize, VARIANT binMin, BSTGR strMin)

Description:

The SDK module notifies the application of this event as a result of GetFpMinutiaeFromTerminal.

Event Parameters

ClientID: 1.6 See term explanation

TerminalID: 1.6 See Terminology Explanation

minType: Feature type (currently only supports 0:UNION Type)

minCount: Number of feature points (currently supports only 2 feature points)

matching: Matching result for the two entered fingerprints (0: Match successful, others: Match failed)

minSize: Length of feature point data (currently 800)

binMin: Feature point data in Hexa Array format (800 bytes)

strMin: Feature point data in hexadecimal string format (1600 bytes)

Reference

GetFpMinutiaeFromTerminal

5. Error definitions

This document defines and explains the various error values used in the UCS SDK.

5.1 Success

This is the definition of the Error value used upon success.

UCSAPIERR_NONE

Prototype:

LONG UCS API ERROR_NONE (0)

Description:

The value returned when the function succeeds. In this case, it indicates the function succeeded, not an error.

5.2 General error definitions

This is the definition of common error values.

UCSAPIERR_INVALID_POINTER

Prototype:

UCSAPIERR_INVALID_POINTER (1)

Description:

An invalid pointer value was used.

UCSAPIERR_INVALID_TYPE

Prototype:

LONG UCS API ERROR: INVALID TYPE (2)

Description:

An incorrect Type value is being used.

UCSAPIERR_INVALID_PARAMETER

Prototype:

LONG UCS API ERROR: INVALID PARAMETER (3)

Description:

An incorrect parameter value was used.

UCSAPIERR_INVALID_DATA

Prototype:

LONG UCS API ERROR: INVALID DATA (4)

Description:

Incorrect data values were used.

UCSAPIERR_FUNCTION_FAIL

Prototype:

LONG UCS API ERROR_FUNCTION_FAIL (5)

Description:

Occurs when a function execution fails due to an internal function error.

UCSAPIERR_NOT_SERVER_ACTIVE

Prototype:

LONG UCSAPIERR_NOT_SERVER_ACTIVE (6)

Description:

The server is not in the initialized state.

UCSAPIERR_INVALID_TERMINAL

Prototype:

LONG UCSAPIERR_INVALID_TERMINAL (7)

Description:

The device is not connected.

UCSAPIERR_PROCESS_FAIL

Prototype:

LONG UCS API ERROR PROCESS FAILURE

(8)

Description:

Failure during processing.

UCSAPIERR_USER_CANCEL

Prototype:

LONG UCSAPIERR_USER_CANCEL

(9)

Description:

Undo by user.

UCSAPIERR_UNKNOWN_REASON

Prototype:

LONG UCSAPIERR_UNKNOWN_REASON

(16)

Description:

Unknown error.

5.3 Data size-related error definitions

This is the definition of the data size error value.

UCS API ERROR CODE SIZE

Prototype:

LONG UCS API ERROR CODE SIZE

(513)

Description:

Access Group Code value size exceeded.

UCSAPIERR_USER_ID_SIZE

Prototype:

LONG UCS API ERROR: User ID Size

(514)

Description:

Maximum size of user ID value exceeded.

UCSAPIERR_USER_NAME_SIZE

Prototype:

LONG UCS API ERROR: USER NAME SIZE (515)

Description:

Maximum size exceeded for the username value.

UCSAPIERR_UNIQUE_ID_SIZE

Prototype:

LONG UCS API ERROR_UNIQUE_ID_SIZE (516)

Description:

Maximum size of UNIQUE ID value exceeded.

UCSAPIERR_INVALID_SECURITY_LEVEL

Prototype:

LONG UCSAPIERR_INVALID_SECURITY_LEVEL (517)

Description:

Certification level value exceeds the range.

UCSAPIERR_PASSWORD_SIZE

Prototype:

LONG UCS API ERROR PASSWORD_SIZE (518)

Description:

Maximum size of user password value exceeded.

UCSAPIERR_PICTURE_SIZE

Prototype:

LONG UCS API ERROR_PICTURE_SIZE (519)

Description:

Maximum size exceeded for user photo image value.

UCSAPIERR_INVALID_PICTURE_TYPE

Prototype:

LONG UCSAPIERR_INVALID_PICTURE_TYPE (520)

Description:

The user photo image type is not supported.

UCSAPIERR_RFID_SIZE

Prototype:

LONG UCSAPIERR_RFID_SIZE (521)

Description:

Maximum size exceeded for the user card number value.

UCSAPIERR_MAX_CARD_NUMBER

Prototype:

LONG UCSAPIERR_MAX_CARD_NUMBER (529)

Description:

The maximum number of cards that can be registered has been exceeded. Up to 5 cards can be registered per user.

UCSAPIERR_MAX_FINGER_NUMBER

Prototype:

LONG UCSAPIERR_MAX_FINGER_NUMBER (530)

Description:

The maximum number of fingerprints that can be registered has been exceeded. Up to 5 fingerprints can be registered per user.

5.4 Authentication-related error definitions

This is the definition of error values related to authentication.

UCSAPIERR_INVALID_USER

Prototype:

LONG UCSAPIERR_INVALID_USER (769)

Description:

Unregistered user.

UCSAPIERR_UNAUTHORIZED

Prototype:

LONG UCSAPIERR_UNAUTHORIZED (770)

Description:

Fingerprint, card, and password matching failed.

UCSAPIERR_PERMISSION

Prototype:

LONG UCSAPIERR_PERMISSION (771)

Description:

No authentication privileges.

UCSAPIERR_FINGER_CAPTURE_FAIL

Prototype:

LONG UCSAPIERR_FINGER_CAPTURE_FAIL (772)

Description:

Fingerprint capture failed.

UCSAPIERR_DUP_AUTHENTICATION

Prototype:

LONG UCSAPIERR_DUP_AUTHENTICATION (773)

Description:

Consecutive authentication attempts. Used to prevent dual authentication on the card when operating in drinking water mode.

UCSAPIERR_ANTIPASSBACK

Prototype:

LONG UCSAPIERR_ANTIPASSBACK (774)

Description:

Anti-passback authentication failed.

UCSAPIERR_NETWORK

Prototype:

LONG UCSAPIERR_NETWORK (775)

Description:

No response from the server due to network issues.

UCSAPIERR_SERVER_BUSY

Prototype:

LONG UCSAPIERR_SERVER_BUSY (776)

Description:

The server is busy and cannot perform authentication.

UCSAPIERR_FACE_DETECTION

Prototype:

LONG UCSAPIERR_FACE_DETECTION (777)

Description:

Face authentication failed when using the face recognition feature.

UCSAPIERR_BLACKLIST

Prototype:

LONG UCSAPIERR_BLACKLIST (778)

Description:

Blacklist authentication failed.

