UNION COMMUNITY
Server SDK For Windows

Programmer's Guide
Version 3.0

June, 2009

Software Development Department

UNION COMMUNITY CO., LTd.

Union Community Co., LTD.
Hyundai Topics Bldg., 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 condition of the agreement, please return the product promptly. If you return the product, you will receive a refund.

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

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

3. Assignment of License
If you wish to transfer the usage license of this software to a third party, you must first obtain a written statement indicating that the recipient agrees with this agreement. You must then transfer the original disk and all other program components, and all copies of the program must be destroyed. After the transfer is complete, you must notify UNION COMMUNITY Co., Ltd. to update the customer registration.
Licensee shall not rent, lease, sell or lend the software application developed using the SDK to a third party without UNION's prior written consent.

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

4. Copyright

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

5. Installation

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

6. Limitation of Warranty

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

7. Liabilities

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

8. Termination

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

9. Customer Service

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

10. General Terms

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

11. Contact

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

Table of Contents

1. Outline 13

1.1 Application 13
1.2 Special Features 13

2. SDK Structure 15

2.1 Development Environment 15
2.3 Module Structure 15
2.4 Terminology Explanation 17

3. Installation 28

3.1 System Requirements 28
3.2 Installing 29
3.3 Installed Files 32
3.3.1 Windows System Directory 32
3.3.2 Executable Module : /Bin 32
3.3.3 Header File : /Include 32
3.3.4 Library File : /lib 33
3.3.5 Sample Source : /Sample 33
3.3.6 Manual : /Manual 33

4. UCSAPI types and macros 34

UCSAPI 34
UCSAPI_BOOL 34
UCSAPI_RETURN 34
UCSAPI_EventHandler 34
UCSAPI_CALLBACK_EVENT 35
UCSAPI_OPTION_FLAG 36
UCSAPI_LOG_TYPE 36
UCSAPI_LENGTH 36
UCSAPI_AUTH_TYPE 37

UCSAPI_AUTH_REQ 37
UCSAPI_AUTH_MODE 38
UCSAPI_PACKET_INFO 38
UCSAPI_DATA 38
UCSAPI_PROCESSING_INFO 39
UCSAPI_VERIFY_INFO 39
UCSAPI_LOCK_SCHEDULE 40
UCSAPI_OPEN_SCHEDULE 41
UCSAPI_TIMEZONE 43
UCSAPI_DAY_SCHEDULE 43
UCSAPI_DAY_SCHEDULE 44
UCSAPI_BASIC_SCHEDULE 44
UCSAPI_EXPAND_SCHEDULE 44
UCSAPI_SECURITY_LEVEL 45
UCSAPI_NETWORK_INFO 45
UCSAPI_SERVER_INFO 46
UCSAPI_TERMINAL_SCHEDULE 46
UCSAPI_BASIC_TERMINAL_OPTIONS 47
UCSAPI_EXPAND_TERMINAL_OPTIONS 48
UCSAPI_TERMINAL_OPTIONS 48
UCSAPI_DATETIME_INFO 49
UCSAPI_LOG_DATA 49
UCSAPI_USER_FLAG 50
UCSAPI_ACCESS_DATE 50
UCSAPI_ACCESS_TIME 51
UCSAPI_USER_INFO 51
UCSAPI_USER_DATA 52
UCSAPI_USER_COUNT 52
UCSAPI_TERMINAL_USER 53
UCSAPI_IPADDRESS 53
UCSAPI_TERMINAL_STATUS 54

5. UCSAPI functions 55

5.1 Initialize Operations 55
UCSAPI_ServerStart 55

UCSAPI_ServerStop 57
5.2 Terminal Database Operations 58
UCSAPI_AddUser 58
UCSAPI_DeleteUser 61
UCSAPI_GetUserCount 63
UCSAPI_GetUserList 64
UCSAPI_GetUserData 65
5.3 Access Log Operations 66
UCSAPI_GetAccessLog 66
UCSAPI_GetAccessLogCount 68
5.4 Authentication Operations 70
UCSAPI_ServerResponse 70
5.5 Terminal Management Operations 72
UCSAPI_GetTerminalCount 72
UCSAPI_GetFirmwareVersion 73
UCSAPI_UpgradeFirmware 74
UCSAPI_GetBasicTerminalOption 76
UCSAPI_GetExpandTerminalOption 77
UCSAPI_SetTerminalOption 78
UCSAPI_TerminalOpen 80
6. UCSCOM method and property 81
6.1 Properties 81
LONG ErrorCode 81
LONG ConnectionsOfTerminal 81
LONG TotalFingerCount 81
LONG FingerID 82
LONG SampleNumber 82
LONG FPSampleData 83
LONG FPSampleDataLength 83
6.2 Methods 85
6.2.1 Initialize Operations 85
ServerStart 85
ServerStop 87
6.2.2 Terminal Database Operations 88

AddMethodPassword(BSTR TextPW) (Addition) 88
AddMethodRFID(BSTR TextRFID) (Addition) 88
AddMethodFinger (Addition) 88
AddUser 89
DeleteUser 92
GetUserCount 94
GetUserList 95
GetUserData 97
6.2.3 Access Log Operations 99
GetAccessLog 99
GetAccessLogCount 101
6.2.4 Terminal Operations 103
GetTerminalCount 103
GetFirmwareVersion 104
UpgradeFirmware 105
6.2.5 Authentication Operations 107
ResponseVerifyInfo 107
ResponseCardVerifyMatch 109
ResponseVerifyMatch 111
ResponseIdentifyMatch 113
6.2.6 Terminal Management Operations 115
OpenTerminal 115
7. Programming 116
7.1 UCSAPI Module Programming 116
7.1.1 Starting and Terminating Server 116
Starting Server 116
Terminating Server 117
7.1.2 Managing Terminal User 118
Adding User 118
Deleting User 119
Acquiring the Number of Users 120
Acquiring User List 121
Acquiring User Information 122
7.1.3 Terminal Log Management 124

Acquiring Log Data 124
Acquiring the Number of Log Data 125
7.1.4 Authenticating Server 127
Responding to User Authentication Type Request 127
Responding to Card Authentication Request 128
Responding to 1:1 Authentication Request 129
Responding to 1:N Fingerprint Authentication Request 131
7.1.5 Managing Terminal 133
Acquiring the Number of Terminals Connected to the Server 133
Acquiring Terminal Firmware Version 133
Upgrading Terminal Firmware 134
Acquiring Terminal Basic Option Value 135
Acquiring Terminal Expanded Option Value 136
Setting Terminal Option Value 137
Forcefully Opening Terminal Lock Device 138
7.2 UCSCOM Module Programming 140
7.2.1 Visual Basic Programming 140
1. Constructing COM Object 140
COM Object Construction 140
COM Object Destruction 140
2. Starting and Terminating Server 141
Starting Server Initialization 141
Terminating Server 142
3. Managing Terminal User 143
Adding User 143
Deleting User 144
Acquiring the Number of Users 145
Acquiring User List 145
Acquiring User Data 146
4. Managing Terminal Log 148
Acquiring Log Data 148
Acquiring the Number of Log Data 149
5. Authenticating Server 150
Responding to User Authentication Type Request 150
Responding to Card Authentication Request 151

Responding to 1:1 Authentication Request 151
Responding to 1:N Fingerprint Authentication Request 152
6. Managing Terminal 154
Acquiring the Number of Terminals Connected to the Server 154
Acquiring Terminal Firmware Version 154
Upgrading Terminal Firmware 155
Acquiring and Setting Terminal Option Value 156
Forcefully Opening Terminal Lock Device 156
7.2.2 Delphi Programming 157
1. Constructing COM Object 157
COM Object Construction 157
COM Object Destruction 157
2. Starting and Terminating Server 158
Starting Server Initialization 158
Terminating Server 158
3. Managing Terminal User 159
Adding User 159
Deleting User 160
Acquiring the Number of Users 160
Acquiring User List 161
Acquiring User Data 161
4. Managing Terminal Log 163
Acquiring Log Data 163
Acquiring the Number of Log Data 164
5. Authenticating Server 165
Responding to User Authentication Type Request 165
Responding to Card Authentication Request 165
Responding to 1:1 Authentication Request 166
Responding to 1:N Fingerprint Authentication Request 167
6. Managing Terminal 168
Acquiring the Number of Terminals Connected to the Server 168
Acquiring Terminal Firmware Version 168
Upgrading Terminal Firmware 168
Acquiring and Setting Terminal Option Value 169
Forcefully Opening Terminal Lock Device 169

8. Error Handling 170

1. Outline

UCS (UNION COMMUNITY Server) SDK was written in high level SDK to develop an application program that can interlock with all network fingerprint recognition terminals of Union Community Co., Ltd.

UCS SDK along with UCB (UNION COMMUNITY Biometric) is intended to provide a server API (Application Programming Interface) and consists of Biometric API for user registration and authentication and server API that can communicate with a network terminal.

1.1 Application

UCS SDK along with UCB SDK defines API on fingerprint recognition technology and server API that can interlock with network terminal products. Therefore, applied in fingerprint authentication system development, it can be used to optimize performance during the fusion and registration of various fingerprint recognition, authentication and recognition.

1.2 Special Features

■ Centralized Management Method

As a method connecting the terminal to UCS module, all terminals are centrally managed and this method has strong advantage in network construction using public network.

■ Various APIs provided for terminal management

UCS SDK provides all APIs to control various functions of the terminal.

■ COM module provided

UCS SDK provides COM base module for C/C++ developers and also developers using Visual Basic or Delphi to make development easy under these tools.

■ Various authentication methods provided

In addition to fingerprint, password, card and various combinations of these are provided as

authentication method to identify users.

2. SDK Structure

2.1 Development Environment

All modules provided in UCS (UNION COMMUNITY Server) SDK were complied under VC++ 6.0 and programming using this SDK is possible under most 32bit compilers such as Visual C++

UCS SDK provides COM module for C/C++ developers and also developers using Visual Basic or Delphi to make development easy under these tools.

2.3 Module Structure

■ Basic Module : UCSAPI.dll

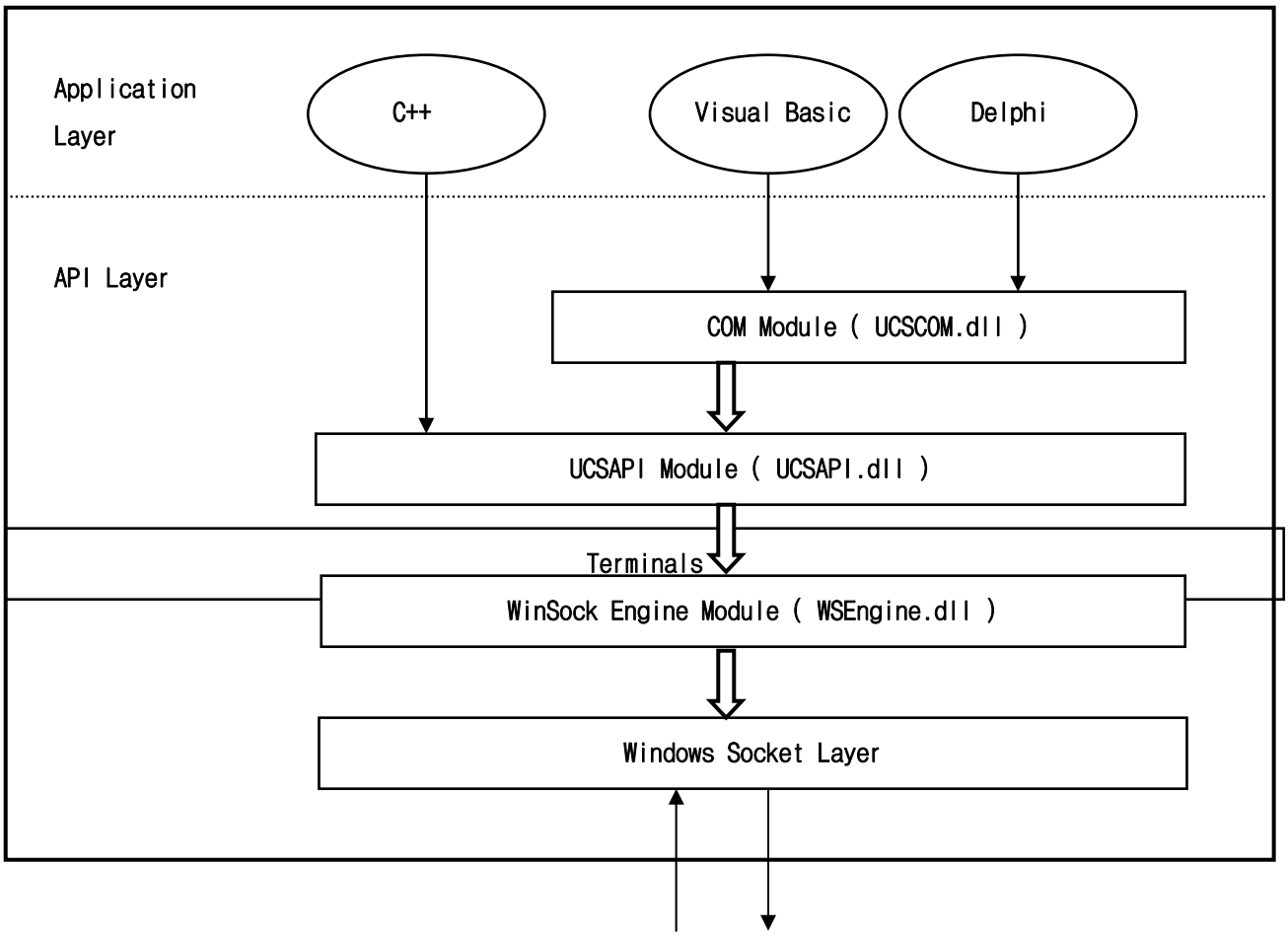
UCSAPI.dll is a standard module to realize function to communicate with network fingerprint recognition terminal.

■ COM Module : UCSCOM.dll

UCSCOM module was developed for developers of RAD tool such as Visual Basic or Delphi. UCSCOM module is written based on UCSAPI module and not all functions of UCSAPI module are provided.

■ Winsock Engine Module : WSEngine.dll

WSEngine.dll WSEngine.dll is a module in charge of Windows Socket IO(Input/Output), and can be replaced according to server capacity and performance.



2.4 Terminology Explanation

■ UCSAPI (UNION COMMUNITY Server API)

It is API module providing function that can communicate with network fingerprint recognition terminals supplied by Union Community Co., Ltd.

■ UCSCOM (UNION COMMUNITY Server COM)

It is a module developed for developers of RAD tool such as Visual Basic or Delphi based on UCSAPI module.

■ Terminal

It is a network fingerprint recognition terminal supplied by Union Community Co., Ltd.

■ Client

It is an application program to communicate with a network fingerprint recognition terminal supplied.

■ Client ID

Client ID is used to develop an application program of client/server model.

Server module requires key to distinguish each client to support multi client environments and t is called clietID.

■ 1:1 Authentication (1 to 1, Verification)

It is handling procedure of 1:1 comparison of fingerprint template (or card, password) corresponding to user ID and presented sample to verify one's identity.

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

It is handling procedure of 1 to N comparison of a part or all fingerprint template and presented sample to verify one's identity.

■ Authentication Request Type

To distinguish users, a fingerprint recognition terminal can use various authentication types upon authentication request to the server.

Request Type	Value(Constant)	Details
1:N fingerprint authentication	0	1:N fingerprint authentication
1:1 fingerprint authentication	1	1:1 fingerprint authentication
Fingerprint card authentication	2	Fingerprint information is stored in smart card and 1:1 authentication between input fingerprint and stored fingerprint (not used)
Card authentication	3	Card authentication
Password authentication	4	Password authentication

■ Authentication Type

The fingerprint recognition terminal supports various authentication types to distinguish users.

Type	Value(Constant)	Details
Fingerprint	0	Fingerprint is used as authentication method.
Fingerprint card	1	Fingerprint card is used as authentication method.
Password	2	Password is used as authentication method.
Card	3	Card is used as authentication method.
Card or fingerprint	4	Card or fingerprint is used as authentication method.
Card and fingerprint	5	The combination of card and fingerprint is used as authentication method.
Card or password	6	Card or password is used as authentication method.
Card and password	7	The combination of card and fingerprint is used as authentication method.
(ID and fingerprint) or (card and fingerprint)	8	The combination of ID and fingerprint or combination of card and fingerprint is used as authentication method.
(ID and password) or (card and password)	9	The combination of ID and password or combination of card and password is used as authentication method.
Fingerprint and password	10	The combination if fingerprint and password is used as authentication.
Fingerprint or password	11	Password is used as authentication method after

		fingerprint authentication failure.

■ Fingerprint Authentication Level (Security Level)

It is level value for successful authentication and its value range is 1~9. UCS SDK recommends developers to use the following level values. The higher the level value is, the higher false rejection rate (FRR) is and the lower false acceptance rate (FAR) is. UCS SDK recommends level 4 for Verification(1:1) authentication and level 5 for Identification(1:N) as standard level. The application can increase authentication level in case of high FAR relative to standard level and decrease authentication level in case of high FRR relative to standard level.

Authentication Level	FMR Value	Description
1	1400 (~ 1800)	Lowest level
2	1800 (~ 2200)	
3	2200 (~ 2600)	
4	2600 (~ 3000)	Recommended level for 1:1 authentication
5	3000 (~ 3500)	Recommended level for 1:N authentication
6	3500 (~ 4000)	
7	4000 (~ 4500)	
8	4500 (~ 5000)	
9	5000 (~ 9999)	Highest level

■ Terminal Working Mode

It defines working method on user's authentication and log record.

Mode	Value (Constant)	Content
N / S	0	The server performs user authentication in general but the terminal performs user authentication in case of network disconnection. Authentication record is stored in the server during server authentication. In case of network disconnection, authentication record is stored in the terminal and authentication record stored in the terminal is transmitted to the server after server connection recovery
S / N	1	User authentication is performed in the terminal in general but authentication request to the server is made for user authentication request unavailable in the terminal. Authentication record is always recorded in the terminal and only authentication result is transmitted to the server when connection between the server and network is established.
NO	2	Only user authentication is performed in the server.
SO	3	Only user authentication is performed in the terminal.

■ Terminal Operation Mode

It defines operation method supported in the terminal.

Mode	Value (Constant)	Content
Access control	0	Terminal is operated as access control mode.
Attendance mode	1	Terminal is operated as attendance management mode.
Drinking water mode	2	Terminal is operated as drinking water management mode.

■ Authentication Mode

It defines authentication purpose during user authentication.

Mode	Vale	Content

	(Constant)	
Coming to office	0	Authentication mode for coming to office
Leaving office	1	Authentication mode for leaving office
General	2	General authentication mode
Working outside	3	Authentication mode for working outside
Return	4	Authentication mode for return

■ Log Type

UCS SDK defines three types of log to acquire log data from the terminal.

Type	Value (Constant)	Content
Log not transmitted to the server	0	New log that has not been transmitted to the server
Log transmitted to the server	1	Log that has already been transmitted to the server
All logs	2	All logs stored in the terminal

■ User Flag

It is a one byte data field that defines authentication type and if a user is administrator.

7 Bit	6 Bit	5 Bit	4 Bit	3 Bit	2 Bit	1 Bit	0 Bit
Admin	Identify	Operation & or |	Card ID	Card	Password	Voice	Fingerprint

User flag can be expressed by the following 12 values.

① Fingerprint

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

② Fingerprint card

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

③ Password

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

④ Card

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

⑤ Card or Fingerprint

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

⑥ Card and Fingerprint

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

⑦ Card or Password

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

⑧ Card and Password

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

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

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

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

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

⑪ Fingerprint and Password

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

⑫ Fingerprint or Password : Password Authentication in case of Fingerprint Authentication Failure

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

■ Terminal Administrator

The terminal administrator represents a user that has authority to change terminal setting information or register/delete user.

■ Terminal State

The terminal transmits its state and the state information of devices attached to the terminal to the server periodically or immediately when there is change in state.

-Terminal state : This value is a state value that changes according to "lock scheduling" function of the terminal.

-Lock device state : This value is a state value that changes according to "open scheduling" function of the terminal.

-Door monitoring state : This value is a door state value that is transmitted from the lock device with monitoring capability.

-Terminal cover state : This value is a value that returns terminal cover state.

State	Value (Constant)	Description
Terminal state	0	Lock cancel state
	1	Lock state (The connection to the server is maintained during terminal lock state but access to the terminal not from the server is not allowed. However, if administrator access permission option is used, only administrator's access is allowed.)
Lock device state	0	Lock state
	1	Open state
Door monitoring state	0	Closed state
	1	Open state
	2	State without monitoring
Terminal cover state	0	Closed state
	1	Open state

■ Callback Event

UCS module uses event method to send data received from the terminal to the application program.

Event	Value (Constant)
UCSAPI_EVENT	1600
UCSAPI_EVENT_CONNECTED	UCSAPI_EVENT+1
UCSAPI_EVENT_DISCONNECTED	UCSAPI_EVENT+2
UCSAPI_EVENT_VERIFY	UCSAPI_EVENT+3
UCSAPI_EVENT_GETACCESSLOG	UCSAPI_EVENT+4
UCSAPI_EVENT_GETACCESSLOGCOUNT	UCSAPI_EVENT+5
UCSAPI_EVENT_ADDUSER	UCSAPI_EVENT+10
UCSAPI_EVENT_DELETEUSER	UCSAPI_EVENT+11
UCSAPI_EVENT_GETUSERCOUNT	UCSAPI_EVENT+12
UCSAPI_EVENT_GETUSERLIST	UCSAPI_EVENT+13
UCSAPI_EVENT_GETUSERDATA	UCSAPI_EVENT+14
UCSAPI_EVENT_REQUEST_VERIFYINFO	UCSAPI_EVENT+20
UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH	UCSAPI_EVENT+21
UCSAPI_EVENT_REQUEST_VERIFYMATCH	UCSAPI_EVENT+22
UCSAPI_EVENT_REQUEST_IDENTIFYMATCH	UCSAPI_EVENT+23
UCSAPI_EVENT_GET_BASIC_TERMINAL_OPTION	UCSAPI_EVENT+30
UCSAPI_EVENT_GET_EXPAND_TERMINAL_OPTION	UCSAPI_EVENT+31
UCSAPI_EVENT_SET_TERMINAL_OPTION	UCSAPI_EVENT+32
UCSAPI_EVENT_FW_UPGRADING	UCSAPI_EVENT+80
UCSAPI_EVENT_FW_UPGRADE	UCSAPI_EVENT+81
UCSAPI_EVENT_FW_VERSION	UCSAPI_EVENT+82
UCSAPI_EVENT_TERMINAL_OPEN	UCSAPI_EVENT+90
UCSAPI_EVENT_TERMINAL_STATUS	UCSAPI_EVENT+91

< Callback Event Summary >

Terminal Connection Notice : UCSAPI_EVENT_CONNECTED

UCS module notifies the application program of UCSAPI_EVENT_CONNECTED event when the terminal is connected.

Terminal Connection Termination Notice : UCSAPI_EVENT_DISCONNECTED

UCS module notifies the application program of UCSAPI_EVENT_DISCONNECTED event when terminal connection is terminated.

Authentication Result Notice : UCSAPI_EVENT_VERIFY

UCS module immediately notifies the application program of UCSAPI_EVENT_VERIFY event when the terminal working in S/N mode and performs user authentication.

Reception Notice to Authentication Log : UCSAPI_EVENT_GETACCESSLOG

UCS module notifies the application program of UCSAPI_EVENT_GETACCESSLOG event in response to the authentication log request of the application program.

Authentication Log Count Notice : UCSAPI_EVENT_GETACCESSLOGCOUNT

UCS module notifies the application program of UCSAPI_EVENT_GETACCESSLOGCOUNT event in response to the authentication log count request of the application program.

Notice of Result on User Addition : UCSAPI_EVENT_ADDUSER

UCS module notifies the application program of UCSAPI_EVENT_ADDUSER event in response to the user addition request of the application program.

Notice of Result on User Deletion : UCSAPI_EVENT_DELETEUSER

UCS module notifies the application program of UCSAPI_EVENT_DELETEUSER event in response to the user deletion request of the application program.

User Count Notice : UCSAPI_EVENT_GETUSERCOUNT

UCS module notifies the application program of UCSAPI_EVENT_GETUSERCOUNT event in response to the user count request of the application program.

User List Notice : UCSAPI_EVENT_GETUSERLIST

UCS module notifies the application program of UCSAPI_EVENT_GETUSERLIST event in response to the user list request of the application program.

User Data Notice : UCSAPI_EVENT_GETUSERDATA

UCS module notifies the application program of UCSAPI_EVENT_GETUSERDATA event in response to the user data request of the application program.

User Authentication Type Request Notice : UCSAPI_EVENT_REQUEST_VERIFYINFO

UCS module notifies the application program of UCSAPI_EVENT_REQUEST_VERIFYINFO event on the user authentication type request of the terminal.

Card User Authentication Request Notice : UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH

UCS module notifies the application program of UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH event on the card user authentication request of the terminal.

User 1:1 Authentication Request Notice : UCSAPI_EVENT_REQUEST_VERIFYMATCH

UCS module notifies the application program of UCSAPI_EVENT_REQUEST_VERIFYMATCH event on the user 1:1 authentication request of the terminal.

User 1:N Authentication Request Notice : UCSAPI_EVENT_REQUEST_IDENTIFYMATCH

UCS module notifies the application program of UCSAPI_EVENT_REQUEST_IDENTIFYMATCH event on the user 1:N authentication request of the terminal.

Terminal Basic Option Information Notice : UCSAPI_EVENT_GET_BASIC_TERMINAL_OPTION

UCS module notifies the application program of UCSAPI_EVENT_GET_BASIC_TERMINAL_OPTION event in response to the terminal basic option information request of the application program.

Terminal Expansion Option Information Notice : UCSAPI_EVENT_GET_EXPAND_TERMINAL_OPTION

UCS module notifies the application program of UCSAPI_EVENT_GET_EXPAND_TERMINAL_OPTION event in response to the terminal expansion option information request of the application program.

Terminal Option Setting Notice : UCSAPI_EVENT_SET_TERMINAL_OPTION

UCS module notifies the application program of UCSAPI_EVENT_SET_TERMINAL_OPTION event in response to the terminal option setting request of the application program.

Firmware Upgrade State Notice : UCSAPI_EVENT_FW_UPGRADING

UCS module notifies the application program of UCSAPI_EVENT_FW_UPGRADING event in response to the firmware upgrade request of the application program.

Firmware Upgrade Completion Notice : UCSAPI_EVENT_FW_UPGRADE

UCS module notifies the application program of UCSAPI_EVENT_FW_UPGRADE event in response to the firmware upgrade request of the application program.

Firmware Version Notice : UCSAPI_EVENT_FW_VERSION

UCS module notifies the application program of UCSAPI_EVENT_FW_VERSION event in response to the firmware version request of the application program.

Notice of Result on Terminal Lock Device Forced Opening : UCSAPI_EVENT_TERMINAL_OPEN

UCS module notifies the application program UCSAPI_EVENT_TERMINAL_OPEN event in response to the user data request of the application program.

Terminal State Notice : UCSAPI_EVENT_TERMINAL_STATUS

UCS module notifies the application program of UCSAPI_EVENT_TERMINAL_STATUS event in response to the user data request of the application program.

3. Installation

3.1 System Requirements

■ CPU

Above Intel Pentium 133Mhz

■ Memory

Above 16M

■ USB Port

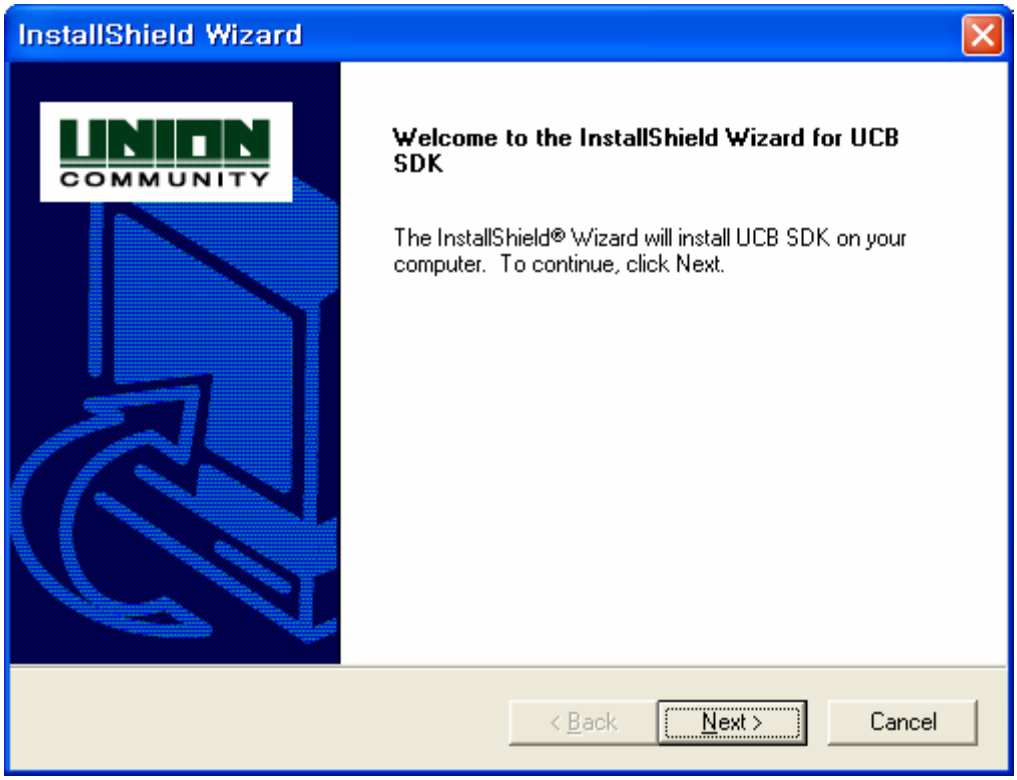
USB 1.1

■ OS

Windows 98/ME or 2000/XP/2003/Vista(32 bit only)

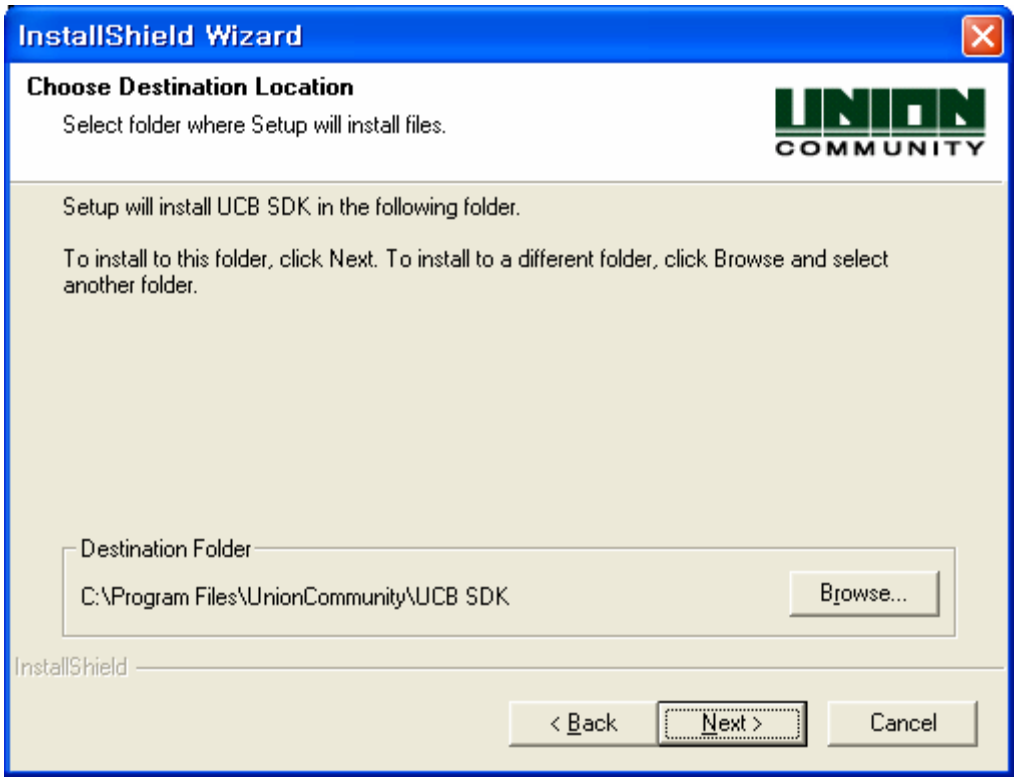
3.2 Installing

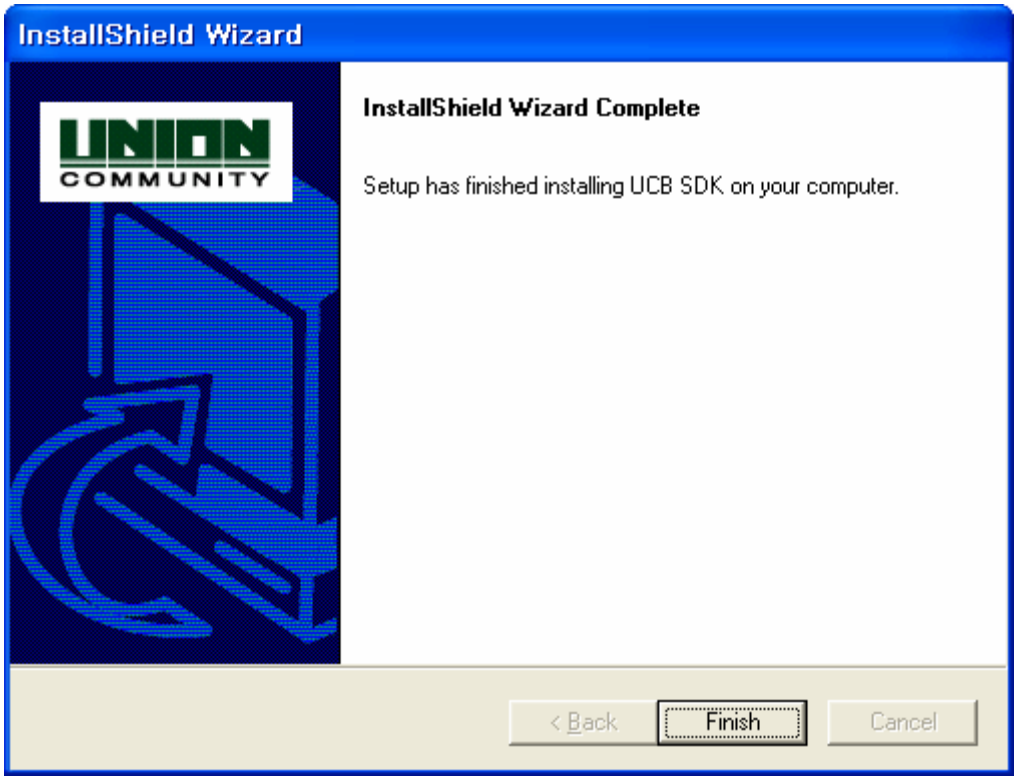
When the installation CD is inserted, Setup.exe starts automatically.



Please check the content in each stage and proceed to install UCS SDK.

Enter user information and product serial number.





3.3 Installed Files

When SDK installation is completed normally, the following files are copied into the designated installation directory.

3.3.1 Windows System Directory

UCSAPI.dll

It is a basic module that realizes all functions related with the network fingerprint recognition terminal.

UCSCOM.dll

COM module for RAD tool developers and web developers

WSEngine.dll

Communication module for processing Windows Socket I/O

3.3.2 Executable Module : /Bin

UCSSample.exe
UCSAPI.dll
UCSCOM.dll
WSEngine.dll

3.3.3 Header File : /Include

ucsapi.h

It is a basic header file of UCSAPI module. The application must include this file to use UCSAPI module. ucsapi.h additionally includes ucsapi_api.h, ucsapi_err.h and ucsapi_type.h file required to use the module.

ucsapi_api.h

The prototype of all functions provided in UCSAPI module is defined.

ucsapi_err.h

Error code used in UCSAPI module is defined.

ucsapi_type.h

The type and macro used in UCSAPI module are defined.

ucsapi_type.h additionally includes ucsapi_porttype.h.

ucsapi_porttype.h

The type and macro used in UCSAPI module are defined.

3.3.4 Library File : /lib

ucsapi.lib

Library file to link the module statically in the application

3.3.5 Sample Source : /Sample

Visual C++

Sample source using UCSAPI.dll

Visual Basic

Sample source using UCSCOM.dll

Delphi

Sample source using UCSCOM.dll

3.3.6 Manual : /Manual

Korean/English manual

4. UCSAPI types and macros

UCSAPI

Definition of the UCSAPI calling conventions

```
#ifdef (WIN32)
#define UCSAPI __stdcall
#else
#define UCSAPI
#endif
```

UCSAPI_BOOL

This type is defined to express the state of success or failure.

```
typedef uint32_t UCSAPI_BOOL;
```

UCSAPI_RETURN

This type is defined to return the handling state of function.

```
typedef uint32_t UCSAPI_RETURN;

        #define UCSAPI_OK     (0)
```

UCSAPI_EventHandler

This type is definition on callback function to receive the event occurred from the terminal.

```
typedef UCSAPI_RETURN   (CALLBACK * UCSAPI_EventHandler) (
        uint32_t TerminalID,
```

```
    uint32_t EventType,
    uint32_t wParam,
    uint32_t lParam);
```

UCSAPI_CALLBACK_EVENT

This type defines the callback event occurred from the terminal.

```
typedef uint32_t UCSAPI_CALLBACK_EVENT;

    #define UCSAPI_EVENT                            1600
    #define UCSAPI_EVENT_CONNECTED                  UCSAPI_EVENT+1
    #define UCSAPI_EVENT_DISCONNECTED               UCSAPI_EVENT+2
    #define UCSAPI_EVENT_VERIFY                     UCSAPI_EVENT+3
    #define UCSAPI_EVENT_GETACCESSLOG               UCSAPI_EVENT+4
    #define UCSAPI_EVENT_GETACCESSLOGCOUNT          UCSAPI_EVENT+5
    #define UCSAPI_EVENT_ADDUSER                    UCSAPI_EVENT+10
    #define UCSAPI_EVENT_DELETEUSER                 UCSAPI_EVENT+11
    #define UCSAPI_EVENT_GETUSERCOUNT               UCSAPI_EVENT+12
    #define UCSAPI_EVENT_GETUSERLIST                UCSAPI_EVENT+13
    #define UCSAPI_EVENT_GETUSERDATA                UCSAPI_EVENT+14
    #define UCSAPI_EVENT_REQUEST_VERIFYINFO         UCSAPI_EVENT+20
    #define UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH    UCSAPI_EVENT+21
    #define UCSAPI_EVENT_REQUEST_VERIFYMATCH UCSAPI_EVENT+22
    #define UCSAPI_EVENT_REQUEST_IDENTIFYMATCH      UCSAPI_EVENT+23
    #define UCSAPI_EVENT_GET_BASIC_TERMINAL_OPTION  UCSAPI_EVENT+30
    #define UCSAPI_EVENT_GET_EXPAND_TERMINAL_OPTION UCSAPI_EVENT+31
    #define UCSAPI_EVENT_SET_TERMINAL_OPTION UCSAPI_EVENT+32
    #define UCSAPI_EVENT_FW_UPGRADING               UCSAPI_EVENT+80
    #define UCSAPI_EVENT_FW_UPGRADE                 UCSAPI_EVENT+81
    #define UCSAPI_EVENT_FW_VERSION                 UCSAPI_EVENT+82
    #define UCSAPI_EVENT_TERMINAL_OPEN              UCSAPI_EVENT+90
```

```
#define UCSAPI_EVENT_TERMINAL_STATUS                    UCSAPI_EVENT+91
```

UCSAPI_OPTION_FLAG

This type defines terminal option flag.

```
typedef uint32_t UCSAPI_OPTION_FLAG;

#define UCSAPI_OPTION_NAME_BIT                  2
#define UCSAPI_OPTION_NAME              1 << UCSAPI_OPTION_NAME_BIT
#define UCSAPI_OPTION_SERVERINFO_BIT            3
#define UCSAPI_OPTION_SERVERINFO      1 << UCSAPI_OPTION_SERVERINFO_BIT
#define UCSAPI_OPTION_NETWORKINFO_BIT           4
#define UCSAPI_OPTION_NETWORKINFO   1 << UCSAPI_OPTION_NETWORKINFO_BIT
#define UCSAPI_OPTION_WORKINGMODE_BIT           6
#define UCSAPI_OPTION_WORKINGMODE 1 << UCSAPI_OPTION_WORKINGMODE_BIT
#define UCSAPI_OPTION_OPERATIONMODE_BIT     7
#define UCSAPI_OPTION_OPERATIONMODE             1                                       <<
UCSAPI_OPTION_OPERATIONMODE_BIT
```

UCSAPI_LOG_TYPE

This type defines type for log operation.

```
typedef uint32_t UCSAPI_LOG_TYPE;

#define UCSAPI_TYPE_NEW_LOG             0
#define UCSAPI_TYPE_OLD_LOG             1
#define UCSAPI_TYPE_ALL_LOG             2
```

UCSAPI_LENGTH

This type defines data length.

typedef uint32_t UCSAPI_LENGTH;

```
#define UCSAPI_LEN_PWD                  8
#define UCSAPI_LEN_CARDNUM              20
```

UCSAPI_AUTH_TYPE

This type defines authentication type.

typedef uint32_t UCSAPI_AUTHTYPE;

```
#define UCSAPI_AUTHTYPE_FP                                  0
#define UCSAPI_AUTHTYPE_FPCARD                              1
#define UCSAPI_AUTHTYPE_PW                                  2
#define UCSAPI_AUTHTYPE_CARD                                3
#define UCSAPI_AUTHTYPE_CARD_OR_FP                          4
#define UCSAPI_AUTHTYPE_CARD_AND_FP                         5
#define UCSAPI_AUTHTYPE_CARD_OR_PW                          6
#define UCSAPI_AUTHTYPE_CARD_AND_PW                         7
#define UCSAPI_AUTHTYPE_ID_AND_FP_OR_CARD_AND_FP            8
#define UCSAPI_AUTHTYPE_ID_AND_PW_OR_CARD_AND_PW            9
#define UCSAPI_AUTHTYPE_FP_AND_PW                           10
#define UCSAPI_AUTHTYPE_FP_OR_PW                            11
```

UCSAPI_AUTH_REQ

This type defines the type designated when the terminal requests authentication to the server.

typedef uint32_t UCSAPI_AUTHREQ;

```
#define UCSAPI_AUTHREQ_FP_1TO1                  0
#define UCSAPI_AUTHREQ_FP_1TON                  1
```

```
#define UCSAPI_AUTHREQ_FPCARD          2
#define UCSAPI_AUTHREQ_CARD            3
#define UCSAPI_AUTHREQ_PW              4
```

UCSAPI_AUTH_MODE

This type defines authentication mode.

```
typedef uint32_t UCSAPI_AUTHMODE;

#define UCSAPI_AUTHMODE_ATTENDANCE     0
#define UCSAPI_AUTHMODE_LEAVE          1
#define UCSAPI_AUTHMODE_NORMAL         2
#define UCSAPI_AUTHMODE_OUT            3
#define UCSAPI_AUTHMODE_RETURN         4
```

UCSAPI_PACKET_INFO

This data type defines information on transmission/reception packet.

```
typedef struct ucsapi_packet_info
{
    uint32_t ClientID;
    uint32_t ErrorCode;
} UCSAPI_PACKET_INFO;
```

ClientID : ID of person requesting work (used in client/server model development)
ErrorCode : Error code on work process result

UCSAPI_DATA

This data type is defined for variable length data.

```
typedef struct ucsapi_data
```

```
{
        uint32_t   Length;
        void*      Data;
} UCSAPI_DATA;
```

Length : Data length
Data : Pointer on data buffer

UCSAPI_PROCESSING_INFO

This type is defined to include information on work progress.

```
typedef struct ucsapi_processing_info
{
        int32_t CurrentBlock;
        int32_t   TotalBlock;
} UCSAPI_PROCESSING_INFO;
```

CurrentBlock : Index of data currently being transmitted
TotalBlock : Total number of data to be transmitted

UCSAPI_VERIFY_INFO

This type is defined to include information on authentication request.

```
typedef struct ucsapi_verify_info
{
        uint32_t UserID;
        int32_t   AuthType;          // 0:FP, 1:FP Card, 2:PW, 3:Card
        int32_t   AuthMode;          // 0:Coming to office, 1:Leaving office, 2:General,
                                     // 3:Working outside, 4:Return
        int32_t   InputIdLength;
        int32_t   SecuLevel;
        uint32_t IsAccessibility;
```

```
    uint32_t IsMatched;
    uint32_t ErrorCode;
    UCSAPI_DATA VerifyData;
} UCSAPI_VERIFY_INFO;
```

UserID : User ID

AuthType : Authentication type

```
    #define UCSAPI_AUTHTYPE_FP          0
    #define UCSAPI_AUTHTYPE_PW          2
    #define UCSAPI_AUTHTYPE_CARD        3
```

AuthMode : Authentication mode

```
    #define UCSAPI_AUTHMODE_ATTENDANCE  0
    #define UCSAPI_AUTHMODE_LEAVE       1
    #define UCSAPI_AUTHMODE_NORMAL      2
    #define UCSAPI_AUTHMODE_OUT         3
    #define UCSAPI_AUTHMODE_RETURN      4
```

InputIdLength : It includes the length of ID pressed through key pad when the terminal requests authentication to the server.

This value can be used to realize shortcut ID function during 1:N server authentication.

If user ID registered in the server is “1234” and ID(UserID) entered from the terminal is “12”with ID length (InputIdLength) entered “2”, the server needs to mark only 99 people from 1200 to 1299.

SecuLevel : Fingerprint authenticating level (Refer to terminology definition.)

IsAccessibility : The terminal includes information if authentication authority exists. If the value is 1, authority exists. If the value is 0, authority does not exist.

IsMatched : It includes authentication execution result. If the value is 1, authority exists. If the value is 0, authority does not exist.

ErrorCode : It includes error code on work processed.

UCSAPIERR_NONE value indicates success and all other values express error condition.

VerifyData : Sample data acquired from the terminal during authentication request. They can be fingerprint sample, password and card number according to AuthType.

UCSAPI_LOCK_SCHEDULE

This type includes scheduling information on terminal lock function.

The terminal can enter lock state or escape from lock state at a scheduled time. Under terminal

lock state, the network connection with terminal logon is maintained but all accesses to the fingerprint recognition terminal except from the server are prohibited.

```
typedef struct ucsapi_lock_schedule
{
        uint16_t Year1;   // Start Year
        uint8_t Month1; // Start Month
        uint8_t Day1;     // Start Day
        uint16_t Year2;   // End Year
        uint8_t Month2; // End Month
        uint8_t Day2;     // End Day
        uint8_t Hour1;   // Start Hour
        uint8_t Min1;     // Start Min
        uint8_t Hour2;   // End Hour
        uint8_t Min2;     // End Min

} UCSAPI_LOCK_SCHEDULE;
```

Year1/Month1/Day1/Hour1/Min1 : It includes the start information of period to enter lock state.
Year2/Month2/Day2/Hour2/Min2 : It includes the terminal information of period to escape from lock state.
If period information is filled with 0, time information will express the start and termination information of lock state during a day.

Note.
If there is no period restriction, fill the values with 0. (YYYYMMDDYYYYMMDD)
If only time is restricted without period restriction, lock function is controlled at a designated time every day.

UCSAPI_OPEN_SCHEDULE

This type includes scheduling information on the opening function of the terminal doorlock.
The terminal can enter doorlock open state or escape from doorlock open state at a scheduled time. Under doorlock open state, access is possible regardless of terminal authentication. This is a state with no control on the terminal doorlock.

```
typedef struct ucsapi_open_schedule
{
        uint16_t Year1;
        uint8_t Month1;
        uint8_t Day1;
        uint16_t Year2;
        uint8_t Month2;
        uint8_t Day2;
        uint8_t Hour1;
        uint8_t Min1;
        uint8_t Hour2;
        uint8_t Min2;

} UCSAPI_OPEN_SCHEDULE;
```

Year1/Month1/Day1/Hour1/Min1 : It includes the start information of period to enter open state.
Year2/Month2/Day2/Hour2/Min2 : It includes the termination information of period to escape from open state.
If period information is filled with 0, time information will express the start and termination information of open state during a day.

Note.
If there is no period restriction, fill the values with 0. (YYYYMMDDYYYYMMDD)
If only time is restricted without period restriction, open function is controlled at a designated time every day.

UCSAPI_TIMEZONE

This type includes time information on time zone.

```
typedef struct ucsapi_timezone
{
        uint8_t IsUsed;
        uint8_t StartHour;
        uint8_t StartMin;
        uint8_t EndHour;
        uint8_t EndMin;

} UCSAPI_TIMEZONE;
```

IsUsed : This includes the validity of time zone. If the value is 1, time zone is valid.
StartHour : Start hour
StartMin : Start minute
EndHour : End hour
EndMin : End minute

UCSAPI_DAY_SCHEDULE

This type includes information on the lock schedule of the terminal and the open schedule of the doorlock by day of the week.
Day of the week scheduling allows up to three lock, open scheduling per day.

```
typedef struct ucsapi_day_schedule
{
        UCSAPI_TIMEZONE Lock1;
        UCSAPI_TIMEZONE Lock2;
        UCSAPI_TIMEZONE Lock3;
        UCSAPI_TIMEZONE Open1;
        UCSAPI_TIMEZONE Open2;
        UCSAPI_TIMEZONE Open3;
```

```
} UCSAPI_DAY_SCHEDULE;
```

UCSAPI_DAY_SCHEDULE

This type includes information on holidays. Holiday is expressed by three types.

```
typedef struct ucsapi_holiday_schedule
{
        int8_t Month;
        int8_t Day;
        int8_t Type;

} UCSAPI_HOLIDAY_SCHEDULE;
```

Month,Day : Date information of holiday set
Number : Holiday type index(1,2,3)

UCSAPI_BASIC_SCHEDULE

This type includes setting information on basic scheduling.

```
typedef struct ucsapi_basic_schedule
{
        UCSAPI_LOCK_SCHEDULE Lock;
        UCSAPI_OPEN_SCHEDULE Open;

} UCSAPI_BASIC_SCHEDULE;
```

UCSAPI_EXPAND_SCHEDULE

This type includes setting information on expanded schedule.
Expanded schedule includes lock schedule and open schedule by day of the week and holidays.
Holidays can be set up to 100 days, its type can be set at Holiday1, Holiday2 and Holiday3.

```
typedef struct ucsapi_expand_schedule
{
        UCSAPI_DAY_SCHEDULE SUN;
        UCSAPI_DAY_SCHEDULE MON;
        UCSAPI_DAY_SCHEDULE TUE;
        UCSAPI_DAY_SCHEDULE WED;
        UCSAPI_DAY_SCHEDULE THU;
        UCSAPI_DAY_SCHEDULE FRI;
        UCSAPI_DAY_SCHEDULE SAT;
        UCSAPI_DAY_SCHEDULE Holiday1;
        UCSAPI_DAY_SCHEDULE Holiday2;
        UCSAPI_DAY_SCHEDULE Holiday3;
        UCSAPI_HOLIDAY_SCHEDULE Holidays[100];

} UCSAPI_EXPAND_SCHEDULE;
```

UCSAPI_SECURITY_LEVEL

This type includes information on fingerprint authentication level.
This value has 1 byte size with upper 4bit Verify(1:1) level information and lower 4bit Identify(1:N) information.

```
typedef struct ucsapi_security_level
{
        uint8_t   Verify          :4;
        uint8_t   Identify  :4;
} UCSAPI_SECURITY_LEVEL;
```

UCSAPI_NETWORK_INFO

This type includes information on terminal network.

```
typedef struct ucsapi_network_info
{
        uint8_t NetworkType;
```

```
        uint8_t   IP[4];
        uint8_t   Subnet[4];
        uint8_t   Gateway[4];
} UCSAPI_NETWORK_INFO;
```

NetworkType : IP type. If this value is 0, it supports fixed IP. If this value is 0, it supports dynamic IP.

IP : Terminal IP information

Subnet : Subnet mask information

Gateway : Gateway information

UCSAPI_SERVER_INFO

This type includes network information for server connection.

```
typedef struct ucsapi_server_info
{
        uint8_t   IP[4];
        int16_t   Port;
} UCSAPI_SERVER_INFO;
```

IP : Server IP.

Port : Port information for server connection

UCSAPI_TERMINAL_SCHEDULE

This includes information on terminal lock and open schedule.
The terminal provides two methods of lock and open scheduling function.
Basic scheduling function can control lock and open during designated period and expanded scheduling function can control lock and open by day of the week.

```
typedef struct ucsapi_terminal_schedule
{
        int32_t ScheduleType; // 0: Basic, 1: Expand
        union
```

```
        {
                UCSAPI_BASIC_SCHEDULE Basic;
                UCSAPI_EXPAND_SCHEDULE Expand;
        }ScheduleData;

} UCSAPI_TERMINAL_SCHEDULE;
```

ScheduleType: The value is 0 or 1

UCSAPI_BASIC_TERMINAL_OPTIONS

This type includes the basic option setting value of the terminal.

```
typedef struct ucsapi_basic_terminal_options
{
        int16_t   Bright;            // 0 - 320
        int16_t   Contrast;          // 0 - 360
        int16_t   Gain;              // 1 - 4
        UCSAPI_SECURITY_LEVEL SecuLevel;
        uint8_t   MicLevel;          // 0 - 7
        uint8_t   IdLength;          // 4 - 8
        uint8_t   IsUseAutoEnter;  // 0/1
        uint8_t   IsUseVoice;        // 0/1
        uint8_t Reserved;
        UCSAPI_TERMINAL_SCHEDULE Schedule;
} UCSAPI_BASIC_TERMINAL_OPTIONS;
```

Bright, Contrast, Gatin : Setting value on fingerprint recognition sensor (This value is not used.)
SecuLevel : Fingerprint authentication level (Refer to Terminology Explanation in section 2.4.)
MicLevel : Microphone volume level (This value is not used.)
IDLength : User ID length
IsUseAutoEnter : It is a value to express terminal's automatic enter function capability. This is a function to input Enter key automatically when there is input as many as ID length.
IsUseVoice : It is a value to express voice use. If the value is 1, voice is used. If the value is 0, Voice is not used.

Reserved : It is a field reserved for future use.
Schedule : Structure including lock and open scheduling information

UCSAPI_EXPAND_TERMINAL_OPTIONS

This type includes the expanded option setting value of the terminal.

```
typedef struct ucsapi_expand_terminal_options
{
        uint8_t   OperationMode;
        uint8_t   WorkingMode;
        uint8_t   Reserved1;
        UCSAPI_NETWORK_INFO NetworkInfo;
        UCSAPI_SERVER_INFO   ServerInfo;
        int8_t    Reserved2[2];
        int8_t    PrintName[32];
} UCSAPI_EXPAND_TERMINAL_OPTIONS;
```

WorkingMode : Terminal working mode (Refer to Terminology Explanation in section 2.4.)
OperationMode : Terminal operation mode (Refer to Terminology Explanation in section 2.4.)
Reserved1 : It is a field reserved for future use.
NetworkInfo : Structure to include network information
ServerInfo : Structure to include server information
Reserved2 : It is a field reserved for future use.
PrintName : Character string to be printed with the printer connected to the terminal

UCSAPI_TERMINAL_OPTIONS

This type is a structure to include terminal option setting value.

```
typedef struct ucsapi_terminal_options
{
        UCSAPI_BASIC_TERMINAL_OPTIONS BasicOptions;
        UCSAPI_EXPAND_TERMINAL_OPTIONS ExpandOptions;
} UCSAPI_TERMINAL_OPTIONS;
```

BasicOptions : Structure to include basic option setting information

ExpandOptions : Structure to include expanded option setting information

UCSAPI_DATETIME_INFO

This type is a structure to include date and time information.

```
typedef struct ucsapi_datetime_info
{
	uint16_t Year;
	uint8_t		Month;
	uint8_t		Day;
	uint8_t		Hour;
	uint8_t		Min;
	uint8_t		Sec;
	uint8_t		Reserved;
} UCSAPI_DATETIME_INFO;
```

UCSAPI_LOG_DATA

This type is a structure to include authentication record.

```
typedef struct ucsapi_log_data
{
	uint32_t UserID;
	UCSAPI_DATETIME_INFO DateTimeInfo;
	uint8_t		AuthMode;
	uint8_t		AuthType;
	uint8_t		IsMatched;
	uint8_t		Reserved;
	UCSAPI_PROCESSING_INFO Processing;

} UCSAPI_LOG_DATA;
```

UserID : User ID

DateTimeInfo : Authentication time

AuthMode : Authentication mode

AuthType : Authentication type

IsMatched : Authentication result. If the value is 1, success. If the value is 0, failure.

Reserved : It is a field reserved for future use.

Processing : It is a structure to express the transmission state of log data.

UCSAPI_USER_FLAG

This type is a structure to include user's basic information.

```
typedef struct ucsapi_user_flag
{
        uint8_t   Finger        :1;
        uint8_t   FPCard        :1;
        uint8_t   Password      :1;
        uint8_t   Card          :1;
        uint8_t CardID          :1;
        uint8_t Operation       :1;
        uint8_t   Identify  :1;
        uint8_t   Admin         :1;
} UCSAPI_USER_FLAG;
```

For detailed explanation on structure members, refer to Terminology Explanation in section 2.4.

UCSAPI_ACCESS_DATE

This is a structure to include information on period accessible to the terminal.
Structure value is used to represent from YYYYMMDD(StartYear/StartMon/StartDay) to YYYYMMDD(EndYear/EndMon/EndDay).

```
typedef struct ucsapi_access_date
{
        uint16_t StartYear;
```

```
    uint8_t     StartMon;
    uint8_t     StartDay;
    uint16_t EndYear;
    uint8_t     EndMon;
    uint8_t     EndDay;
} UCSAPI_ACCESS_DATE;
```

UCSAPI_ACCESS_TIME

This is a structure to include information on time accessible to the terminal.

```
typedef struct ucsapi_access_time
{
    uint8_t     StartHour;
    uint8_t     StartMin;
    uint8_t     EndHour;
    uint8_t     EndMin;
} UCSAPI_ACCESS_TIME;
```

UCSAPI_USER_INFO

This is a structure to include user's authentication authority information.

```
typedef struct ucsapi_user_info
{
    uint32_t        UserID;
    UCSAPI_USER_FLAG    UserFlag;
    UCSAPI_SECURITY_LEVEL       SecuLevel;
    uint8_t             Reserved[2];
    UCSAPI_DATETIME_INFO DateTimeInfo;
    UCSAPI_ACCESS_DATE  AccessDate;
    UCSAPI_ACCESS_TIME  AccessTime;
    uint8_t             Password[UCSAPI_LEN_PWD];
    uint8_t             CardNumber[UCSAPI_LEN_CARDNUM];
```

} UCSAPI_USER_INFO;

UserID : User ID (maximum 8 digits)
UserFlag : UCSAPI_USER_FLAG reference.
SecuLevel : Fingerprint authentication level
Reserved : Field reserved for future use
DateTimeInfo : User registration date
AccessDate : Accessible date
AccessTime : Accessible time (in case AccessDate is filled with 0)
Password : Password information (character string consisting of up to 4 digit number)
CardNumber : RFID information (character string consisting of up to 20 digit hexadecimal number)

Note.
When access authority is net set, AccessDate and AccessTime are filled with 0. If access restriction is set on time without access restriction on period, restriction is effective at a designated time every day.

UCSAPI_USER_DATA

This type is structure to include user's authentication authority information and authentication data.

```
typedef struct ucsapi_user_data
{
        UCSAPI_USER_INFO      UserInfo;
        UCSAPI_DATA           UserData;
} UCSAPI_USER_DATA;
```

UCSAPI_USER_COUNT

This type is a structure to include to the number of use stored in the terminal.
Users are divided into administrator and general users. The terminal administrator has authority to change terminal setting information.

typedef struct ucsapi_user_count

```
{
        uint32_t AdminNumber;
        uint32_t UserNumber;
} UCSAPI_USER_COUNT;
```

AdminNumber : The number of administrators
UserNumber : The number of general users

UCSAPI_TERMINAL_USER

Structure to include information to fetch terminal user list

```
typedef struct ucsapi_terminal_user
{
        UCSAPI_PROCESSING_INFO Processing;
        uint32_t IsAdmin;
        uint32_t UserID;
} UCSAPI_TERMINAL_USER;
```

Processing : It is a structure to express the transmission state of user list.
IsAdmin : Terminal user status. If the value is 1, administrator. If the value is 0, general user.
UserID : User ID.

UCSAPI_IPADDRESS

This type is a structure to include IP address.

```
typedef struct ucsapi_ipaddress
{
        uint8_t addr1;
        uint8_t addr2;
        uint8_t addr3;
        uint8_t addr4;
} UCSAPI_IPADDRESS;
```

UCSAPI_TERMINAL_STATUS

This type is a structure to include terminal state.

```
typedef struct ucsapi_terminal_status
{
        int32_t Terminal;
        int32_t   Door;
        int32_t Cover;
} UCSAPI_TERMINAL_STATUS;
```

For detailed information on member, refer to Terminology Explanation in section 2.4.

5. UCSAPI functions

5.1 Initialize Operations

UCSAPI_ServerStart

```
UCSAPI_RETURN UCSAPI UCSAPI_ServerStart(
        IN uint32_t MaxClient,
        IN uint32_t Port,
        IN int32_t   Reserved,
        IN UCSAPI_EventHandler pUCSAPINotifyCallback);
```

Description

This function initializes UCSAPI module and performs server function.

Parameter

MaxClient: It defines the maximum number of connected terminals. The server adjusts memory use according to the number of clients entered and secures the number of connections by increasing memory use if client connection goes over the maximum number.

Port : Communication port for terminal connection

Reserved : Field reserved for future use

pUCSAPINotifyCallback : Callback function pointer for event notice

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Callback Event

UCSAPI_EVENT_CONNECTED

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : uint8_t ip[4]; // Terminal IP

UCSAPI_ServerStop

UCSAPI_RETURN UCSAPI UCSAPI_ServerStop();

Description

This function cancels the connection of connected terminals and terminates server function.

Parameter

None

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

Callback Event

None

Callback parameter

wParam : UCSAPI_PACKET_INFO

lParam : None

5.2 Terminal Database Operations

UCSAPI_AddUser

```
UCSAPI_RETURN UCSAPI UCSAPI_AddUser(
        IN int32_t   ClientID,
        IN int32_t   TerminalID,
        IN uint32_t UserID,
        IN int8_t * UserName,
        IN uint32_t IsAdmin,
        IN int8_t * DateLimit,
        IN int8_t * TimeLimit,
        IN uint32_t SecuLevel,
        IN uint32_t AuthType,
        IN int8_t * Password,
        IN int8_t * RFID,
        IN UCBioAPI_EXPORT_DATA_PTR* pFingerData);
```

Description

This function downloads user information to the designated terminal.

Previously registered user information will be overwritten.

Parameter

ClientID : Client ID

TerminalID : Terminal ID

UserID: User ID

UserName : Buffer pointer including user name

IsAdmin: Terminal administrator status. If the value is 1, terminal administrator. If the value is 0, general user.

DateLimit: Buffer pointer to set allowed time for authentication. In case of no restriction on authentication period, it is set to NULL value.

TimeLimit: Buffer pointer to set allowed time for authentication. In case of no restriction on

authentication time, it is set to NULL value. In case of no restriction on period, authentication is restricted at a designated time every day.

SecuLevel : Fingerprint authentication level. The range allowed for designation is from 1 to 9. The lower the level value is, the lower false rejection rate (FRR) is and the higher false acceptance rate (FAR) is.

The higher the level value is, the higher false rejection rate (FRR) is and the lower false acceptance rate (FAR) is.

If the level value is 0, the level value designated in the terminal is used.

AuthType: Authentication type. Authentication method can be fingerprint, password, card and their combination.

0 : Fingerprint
1 : Fingerprint card
2 : Password
3 : Card
4 : Card or Fingerprint
5 : Card and Fingerprint
6 : Card or Password
7 : Card and Password

Password : Buffer pointer including password. It is set if AuthType is the combination of passwords. Otherwise, it is set to NULL value.

RFID : Buffer pointer including card number. It is set if AuthType is the combination of cards. Otherwise, it is set to NULL value.

pFingerData : Structure pointer including fingerprint data acquired from UCBioBSP module. It is set if AuthType is the combination of fingerprints. Otherwise, it is set to NULL value.

(For more information, refer to UCBioBSP SDK.)

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_ADDUSER

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : None

※ Cautions

When multiple users are transferred to the terminal using UCSAPI_AddUser function, make sure to call UCSAPI_AddUser and check UCSAPI_EVENT_ADDUSER event if it is processed normal before transferring users.

UCSAPI_DeleteUser

```
UCSAPI_RETURN UCSAPI UCSAPI_DeleteUser(
        IN int32_t   ClientID,
        IN int32_t   TerminalID,
        IN int32_t   UserID);
```

Description

This function deletes user information from the designated terminal.

If UserID value is -1, all user information stored in the terminal is deleted.

Parameter

ClientID : Client ID

TerminalID : Terminal ID

UserID : User ID

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_DELETEUSER

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : None

※ Cautions

When multiple users are deleted from the terminal using UCSAPI_DeleteUser function, make sure to call UCSAPI_DeleteUser and check UCSAPI_EVENT_DELETEUSER event if it is processed normal before deleting users.

UCSAPI_GetUserCount

```
UCSAPI_RETURN UCSAPI UCSAPI_GetUserCount(
        IN int32_t   ClientID,
        IN int32_t   TerminalID);
```

Description

This function acquires the number of registered users from the designated terminal.

Parameter

ClientID : Client ID.

TerminalID : Terminal ID.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_GETUSERCOUNT

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_USER_COUNT

UCSAPI_GetUserList

```
UCSAPI_RETURN UCSAPI UCSAPI_GetUserCount(
        IN int32_t   ClientID,
        IN int32_t   TerminalID);
```

Description

This function acquires the ID list of all registered users from the designated terminal.

Parameter

ClientID : Client ID.

TerminalID : Terminal ID.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_GETUSERLIST

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_TERMINAL_USER

UCSAPI_GetUserData

```
UCSAPI_RETURN UCSAPI UCSAPI_GetUserData(
        IN int32_t   ClientID,
        IN uint32_t TerminalID
        IN uint32_t UserID);
```

Description

This function acquires designated user's data from the designated terminal.

Parameter

ClientID : Client ID.

TerminalID : Terminal ID.

UserID : User ID.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_GETUSERDATA

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_USER_DATA

5.3 Access Log Operations

UCSAPI_GetAccessLog

```
UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLog(
        IN int32_t ClientID,
        IN int32_t TerminalID,
        IN int32_t LogType);
```

Description

This function acquires the authentication log in designated type from the designated terminal.

Parameter

ClientID : Client ID.

TerminalID : Terminal ID.

LogType : Log type

0 : New log that has not been transmitted to the server

1 : Log that has already been transmitted to the server

2 : All logs stored in the terminal

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_GETACCESSLOG

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_LOG_DATA

UCSAPI_GetAccessLogCount

UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLogCount(
IN int32_t ClientID,
IN int32_t TerminalID,
IN int32_t LogType);

Description

This function acquires the number authentication logs in designated type from the designated terminal.

Parameter

ClientID : Client ID.
TerminalID : Terminal ID.
LogType : Log type
0 : Log that has not been transmitted to the server
1 : Log that has already been transmitted to the server
2 : All logs stored in the terminal

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_GETACCESSLOGCOUNT

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : int32_t Count

5.4 Authentication Operations

UCSAPI_ServerResponse

```
UCSAPI_RETURN UCSAPI UCSAPI_ServerResponse(
        IN int32_t TerminalID,
        IN int32_t Message,
        IN int32_t wParam,
        IN int32_t lParam);
```

Description

This function is called to respond to request from the terminal.

During working in N/S (server authentication mode), the terminal can request the application program for user's authentication information to acquire authentication information. Also, the terminal can make authentication request to user's authentication by transmitting entered authentication information (fingerprint, card number and password) to the server.

Parameter

TerminalID : Terminal ID.

Message : Message type on the terminal's request (For more information, refer to Callback Event Summary in section 2.4.)

- UCSAPI_EVENT_REQUEST_VERIFYINFO
- UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH
- UCSAPI_EVENT_REQUEST_VERIFYMATCH
- UCSAPI_EVENT_REQUEST_IDENTIFYMATCH

wParam : It is defined for structure pointer on packet information (UCSAPI_PACKET_INFO).

lParam : It is defined for structure pointer for authentication information (UCSAPI_VERIFY_INFO).

Request Event	Description
UCSAPI_EVENT_REQUEST_VERIFYINFO (User's authentication information request)	The application program must reply to the terminal with authentication type (AuthType) and authentication authority (IsAccessibility). If there is no authentication authority, error on this must also be replied.

UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH (Card number authentication request)	After comparing card number acquired from the terminal with registered information, the application program must reply to the terminal with the result.
UCSAPI_EVENT_REQUEST_VERIFYMATCH (1:1 authentication request)	After comparing authentication information (fingerprint, password) acquired from the terminal with registered information, the application program must reply to the terminal with the result.
UCSAPI_EVENT_REQUEST_IDENTIFYMATCH (1:N fingerprint authentication request)	After comparing fingerprint information acquired from the terminal with registered fingerprints, the application program must reply to the terminal with the result.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_REQUEST_VERIFYINFO

UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH

UCSAPI_EVENT_REQUEST_VERIFYMATCH

UCSAPI_EVENT_REQUEST_IDENTIFYMATCH

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_VERIFY_INFO

5.5 Terminal Management Operations

UCSAPI_GetTerminalCount

```
UCSAPI_RETURN UCSAPI UCSAPI_GetTerminalCount(
        IN int32_t* TerminalCount);
```

Description

This function acquires the number of terminals connected to the server.

Parameter

TerminalCount : Buffer pointer to receive the number of terminals back

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

None

Callback Parameter

None

UCSAPI_GetFirmwareVersion

```
UCSAPI_RETURN UCSAPI UCSAPI_GetFirmwareVersion(
        IN int32_t ClientID,
        IN int32_t TerminalID);
```

Description

This function acquires the firmware version of the designated terminal.

Parameter

ClientID : Client ID.

TerminalID : Terminal ID.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_FW_VERSION

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_DATA

UCSAPI_UpgradeFirmware

```
UCSAPI_RETURN UCSAPI UCSAPI_UpgradeFirmware(
        IN int32_t ClientID,
        IN int32_t TerminalID,
        IN int8_t * FilePath);
```

Description

This function upgrades the designated terminal's firmware.

Parameter

ClientID : Client ID.

TerminalID : Terminal ID.

FilePath : Buffer pointer including firmware file path

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_FW_UPGRADING

UCSAPI_EVENT_FW_UPGRADE

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_PROCESSING_INFO

UCSAPI_GetBasicTerminalOption

```
UCSAPI_RETURN UCSAPI UCSAPI_GetBasicTerminalOption(
        IN int32_t ClientID,
        IN int32_t TerminalID);
```

Description

This function acquires basic option setting value from the terminal.

Details on basic option are defined in UCSAPI_BASIC_TERMINAL_OPTIONS structure.

Parameter

ClientID : Client ID

TerminalID : Terminal ID.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_GET_BASIC_TERMINAL_OPTION

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : UCSAPI_BASIC_TERMINAL_OPTIONS

UCSAPI_GetExpandTerminalOption

UCSAPI_RETURN UCSAPI UCSAPI_GetExpandTerminalOption(
IN int32_t ClientID,
IN int32_t TerminalID);

Description

This function acquires expanded option setting value from the terminal.
Details on expanded option are defined in UCSAPI_EXPAND_TERMINAL_OPTIONS structure.

Parameter

ClientID : Client ID
TerminalID : Terminal ID.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_GET_EVENT_TERMINAL_OPTION

Callback Parameter

wParam : UCSAPI_PACKET_INFO
lParam : UCSAPI_EXPAND_TERMINAL_OPTIONS

UCSAPI_SetTerminalOption

```
UCSAPI_RETURN UCSAPI UCSAPI_SetTerminalOption(
        IN int32_t ClientID,
        IN int32_t TerminalID,
        IN int32_t BasicOptionFlag,
        IN int32_t ExpandOptionFlag,
        IN UCSAPI_BASIC_TERMINAL_OPTIONS* BasicOption,
        IN UCSAPI_EXPAND_TERMINAL_OPTIONS* ExpandOption,
        IN UCSAPI_TERMINAL_SCHEDULE* Schedule);
```

Description

This function changes terminal's option value.

Parameter

ClientID : Client ID

TerminalID : Terminal ID.

BasicOptionFlag : It is the combination value of basic option items to be changed.

- UCSAPI_OPTION_SCHEDULE
- UCSAPI_OPTION_USEVOICE
- UCSAPI_OPTION_AUTOENTER
- UCSAPI_OPTION_IDLENGTH
- UCSAPI_OPTION_MICLEVEL
- UCSAPI_OPTION_SECURITYLEVEL
- UCSAPI_OPTION_SENSOR

ExpandOptionFlag : It is the combination value of expanded option items to be changed.

- UCSAPI_OPTION_SERVERINFO
- UCSAPI_OPTION_NETWORKINFO
- UCSAPI_OPTION_WORKINGMODE
- UCSAPI_OPTION_OPERATIONMODE
- UCSAPI_OPTION_NAME

BasicOption : Pointer on UCSAPI_BASIC_TERMINAL_OPTIONS structure

ExpandOption : Pointer on UCSAPI_EXPAND_TERMINAL_OPTIONS structure

Schedule : Pointer on UCSAPI_TERMINAL_SCHEDULE structure

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_SET_TERMINAL_OPTION

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : None

UCSAPI_TerminalOpen

```
UCSAPI_RETURN UCSAPI UCSAPI_OpenTerminal(
        IN int32_t ClientID,
        IN int32_t TerminalID);
```

Description

This function forcefully opens the lock device attached to the terminal.

Parameter

ClientID : Client ID

TerminalID : Terminal ID.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback Event

UCSAPI_EVENT_TERMINAL_OPEN

Callback Parameter

wParam : UCSAPI_PACKET_INFO

lParam : None

6. UCSCOM method and property

6.1 Properties

Various properties of UCSCOMObj Object are described.

LONG ErrorCode

Prototype:

[ReadOnly] long ErrorCode;

Description:

Values for errors occurred during the setting of executed methods and properties are stored.

The value of 0 means success and all other values mean failure.

Errors occurred during the setting of the method and property of a child object can also be obtained using this value

LONG ConnectionsOfTerminal

Prototype:

[ReadOnly] long ConnectionsOfTerminal;

Description:

It includes the number of terminals connected to the server.

This value can be acquired after calling GetTerminalCount() method.

LONG TotalFingerCount

Prototype:

[ReadOnly] long TotalFingerCount;

Description:

The total number of fingers of converted FIR is stored.

It is used only after the EventGetUserData is called.

Related methods:

EnrollFromTerminal

Related properties:

FingerID

LONG FingerID

Prototype:

[ReadOnly] long FingerID(long nIndex);

Description:

The finger ID information of converted FIR is stored as array.

nIndex can have a value ranging in 0 ~ (TotalFingerCount – 1).

It is used only after the EventGetUserData is called.

Related methods:

EnrollFromTerminal

Related properties:

FPSampleDataLength, FPSampleData

LONG SampleNumber

Prototype:

[ReadOnly] long SampleNumber;

Description:

The number of finger-by-finger templates of converted FIR is stored. The value of either 1 or 2 is stored. It is used only after the EventGetUserData is called.

Related methods:

EnrollFromTerminal

Related properties:

FPSampleDataLength, FPSampleData

LONG FPSampleData

Prototype:

```
[ReadOnly]    VARIANT        FPSampleData(
                                 long nFingerID,
                                 long SampleNum);
```

Description:

Binary stream data of finger-by-finger templates of converted FIR are obtained.

nFingerID and SampleNum can be obtained using FingerID and SampleNumber property.

It is used only after the EventGetUserData is called.

Parameters:

nFingerID:

The ID of a finger to be obtained.

nSampleNumber:

The number of a sample to be obtained. The value of either 0 or 1 is used.

Related methods:

EnrollFromTerminal

Related properties:

FingerID, SampleNumber, FPSampleDataLength

LONG FPSampleDataLength

Prototype:

```
[ReadOnly]    long           FPSampleDataLength(
                                 long nFingerID,
                                 long SampleNum);
```

Description:

The data size of finger-by-finger template of converted FIR is obtained.
nFingerID and SampleNum can be obtained using FingerID 및 SampleNumber property.
It is used only after the EventGetUserData is called.

Parameters:

nFingerID:

The ID of a finger to be obtained.

nSampleNumber:

The number of a sample to be obtained. The value of either 0 or 1 is used.

Related methods:

EnrollFromTerminal

Related properties:

FingerID, SampleNumber, FPSampleData

6.2 Methods

6.2.1 Initialize Operations

ServerStart
(LONG MaxClient,
  LONG Port)

Description
This method initializes UCSCOM module and performs server function.

Parameter
MaxClient: It defines the maximum number of terminals that can be connected. The server adjusts internal memory use according to the number of clients entered and secures the number of connections by increasing memory automatically if client connections go over the maximum number.
Port : Communication port for terminal connection

Return Value
UCSAPIERR_NONE value indicates success. All other values express error condition.

Error
UCSAPIERR_FUNCTION_FAILED
UCSAPIERR_NOT_ACTIVE
UCSAPIERR_START_SERVER

Related Properties
ErrorCode

Event
EventTerminalConnected(LONG TerminalID, BSTR TerminalIP)

Event Parameter
TerminalID: Terminal ID

TerminalIP: Terminal IP string

ServerStop

Description

This method cancels the connection of terminals connected and terminates server function.

Parameter

None

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

None

Event Parameter

None

6.2.2 Terminal Database Operations

AddMethodPassword(BSTR TextPW) (Addition)

Definition

This method is for adding password info to the user and this has to be called before AddUser method.

Parameter

TextPW : Buffer pointer containing password info. It applies only when AuthType is password combination.

AddMethodRFID(BSTR TextRFID) (Addition)

Definition

This method is for adding RFID info to the user and this has to be called before AddUser method.

Parameter

TextRFID : Buffer pointer containing RFID info. It applies only when AuthType is RFID combination.

AddMethodFinger (Addition)

```
( BOOL bInitiialize,
  LONG nSrcFPDataType,
  LONG nFPDataSize,
  VARIANT FPData1,
  VARIANT FPData2
)
```

definition

This method is for adding FP info to the user and this has to be called before AddUser method.

Parameter

bInitialize : It initialize FIR data and defines if it needs to make a new one.

If this value is False, newly adding template data will be continuously added to FIR data which is made internally and there will be one FIR data which contains multiple template data. If this

value is True, it will delete the existing FIR and makes a new one.

nSrcFPDataType:
Adding Template Type Info. Refer to UCBioAPI_TEMPLATE_TYPE for related value.

FPData1:
Adding Template data. (Binary stream data)

FPData2:
Adding the second fingeprirnt Template data. (Binary stream data)
This value is not needed for defining as option. If so, SamplesPerFinger value for FIR to be generated is 1 internally.

AddUser
(LONG ClientID,
  LONG TerminalID,
  LONG UserID,
  BSTR UserName,
  LONG IsAdmin,
  BSTR DateLimit,
  BSTR TimeLimit,
  LONG SecuLevel,
  LONG AuthType,
  BSTR TextPW,
  BSTR TextRFID,
  BSTR TextBIR)

Description
This method downloads user information from the designated terminal.
Previously registered user information will be overwritten.

Parameter
ClientID : Client ID
TerminalID : Terminal ID
UserID: User ID

UserName : Buffer pointer including user name

IsAdmin: Terminal administrator status. If the value is 1, terminal administrator. If the value is 0, general user.

DateLimit: Buffer pointer to set allowed period for authentication. If authentication period is not restricted, it is set to NULL value.

TimeLimit: Buffer pointer to set allowed time for authentication. If authentication time is not restricted, it is set to NULL value. If period is not restricted, authentication is restricted at a designated time every day.

SecuLevel : Fingerprint authentication level. The range allowed for designation is from 1 to 9. The lower the level value is, the lower false rejection rate (FRR) is and the higher false acceptance rate (FAR) is.

The higher the level value is, the higher false rejection rate (FRR) is and the lower false acceptance rate (FAR) is.

If the level value is 0, the level value designated in the terminal is used.

AuthType: Authentication type. Authentication method can be fingerprint, password, card and their combination.

0 : Fingerprint
1 : Fingerprint card
2 : Password
3 : Card
4 : Card or Fingerprint
5 : Card and Fingerprint
6 : Card or Password
7 : Card and Password

Password : Buffer pointer including password. It is set if AuthType is the combination of passwords. Otherwise, it is set to NULL value.

RFID : Buffer pointer including card number. It is set if AuthType is the combination of cards. Otherwise, it is set to NULL value.

BDB : Buffer pointer including fingerprint data acquired from UCBioBSP module. It is set if AuthType is the combination of fingerprints. Otherwise, it is set to NULL value.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventAddUser(LONG Client, LONG TerminalID);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

※ Cautions

When multiple users are transferred to the terminal using AddUser method, make sure to call AddUser and check EventAddUser event if it is processed normal before transferring users.

DeleteUser

(LONG ClientID,

  LONG TerminalID,

  LONG UserID)

Description

This method deletes user data from the designated terminal.

If UserID value is –1, all user information stored in the terminal is deleted.

Parameter

ClientID : Client ID

TerminalID : Terminal ID

UserID: User ID

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventDeleteUser(LONG Client, LONG TerminalID);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

※ Cautions

When multiple users are transferred to the terminal using DeleteUser meothod, make sure to call DeleteUser and check EventDeleteUser event if it is processed normal before transferring users.

GetUserCount

(LONG ClientID,

  LONG TerminalID)

Description

This method acquires the number of registered users from the designated terminal.

Parameter

ClientID : Client ID

TerminalID : Terminal ID

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventGetUserCount(LONG ClientID, LONG TerminalID, LONG AdminCount, LONG UserCount);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

AdminCount : The number of terminal administrators

UserCount : The number of general users

GetUserList
(LONG ClientID,
  LONG TerminalID)

Description
This method acquires the ID list of all registered users from the designated terminal.

Parameter
ClientID : Client ID
TerminalID : Terminal ID

Return Value
UCSAPIERR_NONE indicates success. All other values express error condition.

Error
UCSAPIERR_FUNCTION_FAILED
UCSAPIERR_NOT_ACTIVE
UCSAPIERR_START_SERVER

Related Properties
ErrorCode

Event
EventGetUserList(LONG ClientID, LONG TerminalID, LONG UserID, LONG IsAdmin, LONG CurrentBlock, LONG TotalBlock);

Event Parameter
ClientID : Client ID
TerminalID : Terminal ID
UserID : User ID
IsAdmin : Terminal administrator status (If the value is 1, terminal administrator. If the value is 0, general user.)
CurrentBlock : Index of block data currently being transmitted

TotalBlock : The number of total block data to be transmitted

GetUserData
(LONG ClientID,
  LONG TerminalID,
  LONG UserID)

Description

This method acquires designated user's data from the designated terminal.

Parameter

ClientID : Client ID

TerminalID : Terminal ID

UserID : User ID

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

FPSampleData

TotalFingerCount

FingerID

FPSampleDataLength

SampleNumber

Event

EventGetUserData(LONG ClientID, LONG TerminalID, LONG UserID, BSTR UserName, LONG IsAdmin, BSTR DateLimit, BSTR TimeLimit, LONG SecuLevel, LONG AuthType, BSTR TextPW, BSTR TextRFID);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

UserID : User ID

UserName : User name

IsAdmin: Terminal administrator status. If the value is 1, terminal administrator. If the value is 0, general user.

DateLimit: Allowed period value for authentication. In case of no restriction on authentication period, it is set to NULL value.

TimeLimit: Allowed time value for authentication. In case of no restriction on authentication time, it is set to NULL value. In case of no restriction on period, authentication is restricted at a designated time every day.

SecuLevel : Fingerprint authentication level. Allowed range for designation is from 1 to 9. The lower the level value is, the lower false rejection rate (FRR) is and the higher false acceptance rate (FAR) is.

AuthType: Authentication type. Authentication method can be fingerprint, password, card and their combination.

0 : Fingerprint

1 : Fingerprint card

2 : Password

3 : Card

4 : Card or Fingerprint

5 : Card and Fingerprint

6 : Card or Password

7 : Card and Password

TextPW : Buffer pointer including password. It is set if AuthType is the combination of passwords. Otherwise, it is set to NULL value.

TextRFID : Buffer pointer including card number. It is set if AuthType is the combination of cards. Otherwise, it is set to NULL value.

6.2.3 Access Log Operations

GetAccessLog
(LONG ClientID,
  LONG TerminalID,
  LONG LogType)

Description
This method acquires designated user's data from the designated terminal.

Parameter
ClientID : Client ID
TerminalID : Terminal ID
LogType : Authentication log type
  0 : New log that has not been transmitted to the server
  1 : Log that has already transmitted to the server
  2 : All logs stored in the terminal

Return Value
UCSAPIERR_NONE value indicates success. All other values express error condition.

Error
UCSAPIERR_FUNCTION_FAILED
UCSAPIERR_NOT_ACTIVE
UCSAPIERR_START_SERVER

Related Properties
ErrorCode

Event
EventGetAccessLog(LONG ClientID, LONG TerminalID, LONG UserID, BSTR AccessTime, LONG AuthMode, LONG AuthType, LONG IsMatched, LONG CurrentBlock, LONG TotalBlock);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

UserID : User ID

AccessTime : Authentication time

AuthMode : Authentication mode

AuthType : Authentication type

0 : 1:1 fingerprint

1 : 1:N fingerprint

2 : Password

3 : Card

IsMatched : Authentication result

CurrentBlock : Index of Block currently being transmitted

TotalBlock : The number of total blocks to be transmitted.

GetAccessLogCount
(LONG ClientID,
  LONG TerminalID,
  LONG LogType)

Description

This method acquires the number of authentication logs in designated type from the designated terminal.

Parameter

ClientID : Client ID

TerminalID : Terminal ID

LogType : Authentication log type

  0 : New log that has not been transmitted to the server

  1 : Log that has already been transmitted to the server

  2 : All logs stored in the terminal

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventGetAccessLogCount(LONG ClientID, LONG TerminalID, LONG LogCount);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

LogCount : The number of logs

6.2.4 Terminal Operations

GetTerminalCount

Description

This method acquires the number of terminals connected to the server.

Parameter

None

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

TerminalCount

Event

None

Event Parameter

None

GetFirmwareVersion
(LONG ClientID,
  LONG TerminalID)

Description

This method acquires the firmware version of the designated terminal.

Parameter

ClientID : Client ID

TerminalID : Terminal ID

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventFirmwareVersion(LONG ClientID, LONG TerminalID, BSTR Version);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

Version : Firmware Version

UpgradeFirmware
(LONG ClientID,
  LONG TerminalID,
  BSTR FilePath)

Description

This method upgrades the firmware of the designated terminal.

Parameter

ClientID : Client ID.

TerminalID : Terminal ID.

FilePath : String including firmware file path

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventFirmwareUpgrading(LONG ClientID, LONG TerminalID, LONG CurrentBlock, LONG TotalBlock);

EventFirmwareUpgrade(LONG ClientID, LONG TerminalID);

Event Parameter

ClientID : Client ID

TerminalID : Terminal ID

CurrentBlock : Index of block data currently being transmitted

TotalBlock : The number of total block data to be transmitted

6.2.5 Authentication Operations

ResponseVerifyInfo
(LONG TerminalID,
  LONG UserID,
  LONG AuthType,
  LONG IsAccessibility,
  LONG ErrorCode)

Description

This method is called to return result on user authentication request.

The application program transmits authentication information of the user designated from the terminal to the terminal.

This method is called for the application program to reply with user's authentication type (AuthType) and authentication authority (IsAccessibility) requested from the terminal to the terminal. In case of no authentication authority, error on this must also be replied.

Parameter

TerminalID : Terminal ID.

UserID : User ID

AuthType : Authentication type

IsAccessibility : Authentication authority status. If the value is 1, authentication authority exists. If the value is 0, authentication authority does not exist.

ErrorCode : It includes error on authentication authority.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventRequestVerifyInfo(LONG TerminalID, LONG UserID);

Event Parameter

TerminalID : Terminal ID

UserID : User ID

ResponseCardVerifyMatch
(LONG TerminalID,
  LONG UserID,
  LONG IsAccessibility,
  LONG IsMatched,
  LONG ErrorCode)

Description

This method is called to return result on card authentication request.

After comparing card data acquired from the terminal with registered information, the application program reply with the result to the terminal.

Parameter

TerminalID : Terminal ID.

UserID : User ID.

IsAccessibility : Authentication authority status. If the value is 1, authentication authority exists. If the value is 0, authentication authority does not exist.

IsMatched : Authentication result. If the value is 1, authentication authority exists. If the value is 0, authentication authority does not exist.

ErrorCode : It includes error on authentication authority and result.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventRequestCardVerifyMatch(LONG TerminalID, LONG AuthMode, BSTR TextRFID);

Event Parameter

TerminalID : Terminal ID

AuthMode : Authentication mode

TextRFID : Text type RFID string

ResponseVerifyMatch
(LONG TerminalID,
  LONG UserID,
  LONG IsAccessibility,
  LONG IsMatched,
  LONG ErrorCode)

Description

This method is called to return result on 1:1 authentication request.

After comparing fingerprint or password data acquired from the terminal with registered information, the application program replies with result to the terminal.

Parameter

TerminalID : Terminal ID.

UserID : User ID.

IsAccessibility : Authentication authority status. If the value is 1, authentication authority exists. If the value is 0, authentication authority does not exist.

IsMatched : Authentication result. If the value is 1, authentication authority exists. If the value is 0, authentication authority does not exist.

ErrorCode : It includes error on authentication authority and result.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED

UCSAPIERR_NOT_ACTIVE

UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventRequestVerifyMatch(LONG TerminalID, LONG UserID, LONG AuthMode, LONG SecuLevel, LONG AuthType, BSTR VerifyData);

Event Parameter

TerminalID : Terminal ID

UserID : User ID

AuthMode : Authentication mode

UCSAPI_AUTHMODE_ATTENDANCE	0
UCSAPI_AUTHMODE_LEAVE	1
UCSAPI_AUTHMODE_NORMAL	2
UCSAPI_AUTHMODE_OUT	3
UCSAPI_AUTHMODE_RETURN	4

SecuLevel : Fingerprint authentication level (Refer to Terminology Explanation.)

AuthType : Authentication type

UCSAPI_AUTHTYPE_FP	0
UCSAPI_AUTHTYPE_PW	2
UCSAPI_AUTHTYPE_CARD	3

VerifyData : Data according to authentication type (fingerprint or password)

ResponseIdentifyMatch
(LONG TerminalID,
  LONG UserID,
  LONG IsAccessibility,
  LONG IsMatched,
  LONG ErrorCode)

Description

This method is called to transmit result on 1:N authentication request to the terminal.
After comparing fingerprint data acquired from the terminal with registered information, the application program replies with the result to the terminal.

Parameter

TerminalID : Terminal ID
UserID : Authenticated user ID
IsAccessibility : Authentication authority status. If the value is 1, authentication authority exists. If the value is 0, authentication authority does not exist.
IsMatched : Authentication result. If the value is 1, authentication authority exists. If the value is 0, authentication authority does not exist.
ErrorCode : It includes error on authentication authority and result.

Return Value

UCSAPIERR_NONE value indicates success. All other values express error condition.

Error

UCSAPIERR_FUNCTION_FAILED
UCSAPIERR_NOT_ACTIVE
UCSAPIERR_START_SERVER

Related Properties

ErrorCode

Event

EventRequestIdentifyMatch(LONG TerminalID, LONG AuthMode, LONG InputIDLength, LONG SecuLevel, LONG AuthType, BSTR VerifyData);

Event Parameter

TerminalID : Terminal ID

AuthMode : Authentication mode

InputIDLength : It includes the length of ID pressed through key pad during authentication request from the terminal to the server.

This value can be used to realize shortcut ID function during 1:N server authentication.

If user ID registered in the server is “1224” and input ID (UserID) from the terminal is “12” with the length of input ID (InputIdLength) “2”, the server needs to mark only 99 people from 1200 to 1299.

SecuLevel : Fingerprint authentication level (Refer to Terminology Explanation.)

AuthType : Authentication type acquired at the terminal

VerifyData : Sample data acquired from the terminal during authentication request. They can be fingerprint sample, password or card number according to AuthType.

6.2.6 Terminal Management Operations

OpenTerminal
(LONG ClientID,
  LONG TerminalID)

Description
This method forcefully opens the lock device attached to the terminal.

Parameter
ClientID : Client ID
TerminalID : Terminal ID.

Return Value
UCSAPIERR_NONE value indicates success. All other values express error condition.

Error
UCSAPIERR_FUNCTION_FAILED
UCSAPIERR_NOT_ACTIVE
UCSAPIERR_START_SERVER

Related Properties
ErrorCode

Event
EventOpenTerminal(LONG ClientID, LONG TerminalID);

Event Parameter
ClientID : Client ID
TerminalID : Terminal ID

7. Programming

7.1 UCSAPI Module Programming

7.1.1 Starting and Terminating Server

The application program must call UCAPI_ServerStart() function to start server function. UCAPI_ServerStart() function puts terminal's connection to the designated connection port always on standby. Also, the application program must call UCSAPI_ServerStop function to terminate server function.

Starting Server

```
typedef UCSAPI_RETURN  (CALLBACK * UCSAPI_EventHandler) (
        uint32_t TerminalID,
        uint32_t EventType,
        uint32_t wParam,
        uint32_t lParam);

LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        switch(EventType)
        {
        case UCSAPI_EVENT_CONNECTED:
             break;
        case UCSAPI_EVENT_DISCONNECTED:
             break;
         .
```

```
        .
        .
       }
}

LONG MaxClient = 1;
LONG Port = 2201;
UCSAPI_RETURN ret = UCSAPI_ServerStart(MaxClient,
                                       Port,
                                       0,
                                       (UCSAPI_EventHandler) UCSAPINotifyCallback);
If(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail
```

Terminating Server

The application program must call UCSAPI_ServerStop() function to terminate server function.

```
UCSAPI_RETURN ret = UCSAPI_ServerStop();
If(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail
```

7.1.2 Managing Terminal User

UCSAPI module provides function to add or delete terminal users by the application program.

Adding User

The application program calls UCSAPI_AddUser() function to add user information to the terminal.

```
uint32_t AuthType = UCSAPI_AUTHTYPE_FP;
uint32_t SecuLevel = 0; //If authentication level is 0, the terminal uses terminal's //authentication
                        level value. To store user's authentication level //separately, the
                        value must be other than 0.
uint32_t UserID = 1; // User ID can range from 1 to 99999999.
UCSAPI_BOOL IsAdmin = FALSE;
int8_t UserName[32] = {0,};
int8_t DateLimit[16] = {0,}; //"YYYYMMDDYYYYMMDD"
                        //It is 16 byte character string to restrict authentication on period.
                        //In case of no authentication restriction on period, the value is
                        //filled with 0.

int8_t* TimeLimit[8] = {0,}; //"HHMMHHMM"
                        //It is 8 byte character string to restrict authentication on time.
                        //In case of no authentication restriction on time, the value is
                        //filled with 0.

UCSAPI_RETURN ret = UCSAPI_AddUser(
    m_ClientID,
    TerminalID,
    UserID,
    (int8_t *)UserName,
    IsAdmin,
    (int8_t *)DateLimit, //In case of no authentication restriction on period, NULL can be
                        //entered.
    (int8_t *)TimeLimit, //In case of no authentication restriction on time, NULL can be
                        entered.
    SecuLevel,
    AuthType,
    pszPassword,
    pszRFID,
    (UCBioAPI_EXPORT_DATA_PTR)pFingerData);


if(ret == UCSAPIERR_NONE)
    //Success
```

```
else
    //Fail


// Callback event on UCSAPI_AddUser
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        switch(EventType)
        {
        case UCSAPI_EVENT_ADDUSER:
            if(pi.ErrorCode == 0)
                // Success
            else
                // Fail
            break;
        }
}
```

Deleting User

The application program calls UCSAPI_DeleteUser() function to delete user from the terminal.

To delete all users in the terminal, set UserID to -1.

```
UCSAPI_RETURN ret = UCSAPI_DeleteUser(
    m_ClientID,
    TerminalID,
    UserID);


if(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail
```

```
// Callback event on UCSAPI_DeleteUser
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        switch(EventType)
        {
        case UCSAPI_EVENT_DELETEUSER:
            if(pi.ErrorCode == 0)
                // Success
            else
                // Fail
            break;
        }
}
```

Acquiring the Number of Users

To acquire the number of administrators and general users registered in the terminal, the application program calls UCSAPI_GetUserCount() function.

```
UCSAPI_RETURN ret = UCSAPI_GetUserCount(m_ClientID, TerminalID);

if(ret == UCSAPIERR_NONE)
      //Success
else
      //Fail

// Callback event on UCSAPI_ GetUserCount
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
```

```
        pi = (UCSAPI_PACKET_INFO*)wParam;
        UCSAPI_USER_COUNT UserCount;
        switch(EventType)
        {
        case UCSAPI_EVENT_GETUSERCOUNT:
            if(pi.ErrorCode == 0)
                // Success
                memcpy(&UserCount,                    (UCSAPI_USER_COUNT*)lParam,
sizeof(UCSAPI_USER_COUNT));
            else
                // Fail
            break;
        }
  }
```

Acquiring User List

To acquire ID list information of users registered in the terminal, the application program calls UCSAPI_GetUserList() function.

```
  ret = UCSAPI_GetUserList(m_ClientID, TerminalID);

  if(ret == UCSAPIERR_NONE)
      //Success
  else
      //Fail

  // Callback event on UCSAPI_ GetUserList
  LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
  {
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        UCSAPI_TERMINAL_USER Users;
        switch(EventType)
        {
```

```
        case UCSAPI_EVENT_GETUSERLIST:
            if(pi.ErrorCode == 0)
                // Success
                memcpy(&Users,                                    (UCSAPI_TERMINAL_USER*)lParam,
sizeof(UCSAPI_TERMINAL_USER));
            else
                // Fail
            break;
        }
  }
```

Acquiring User Information

To acquire user information registered in the terminal, the application program calls UCSAPI_GetUserData() function.

```
  UCSAPI_RETURN ret = UCSAPI_GetUserData(m_ClientID, TerminalID, UserID);


  if(ret == UCSAPIERR_NONE)
      //Success
  else
      //Fail


  // Callback event on UCSAPI_ GetUserData
  LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
  {
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        UCSAPI_USER_DATA UserData;
        switch(EventType)
        {
        case UCSAPI_EVENT_GETUSERDATA:
            if(pi.ErrorCode == 0)
                // Success
                memcpy(&UserData, (UCSAPI_USER_DATA*)lParam, sizeof(UCSAPI_USER_DATA));
            else
                // Fail
```

```
            break;
        }
    }
```

7.1.3 Terminal Log Management

The terminal has different storing method according to working mode.
First, authentication log is stored in the application program when the terminal works in N/S mode. Only if the application fails to store authentication, it is stored in the terminal. This method depends on the application program for most of authentication work, and log recovery becomes difficult when application program's databases are damaged.
On the other hand, authentication log is stored in the terminal and simultaneously is transmitted to the application program when the terminal works in S/N mode. This method reduces the danger of losing log by storing authentication log in two places.

Acquiring Log Data

UCSAPI_GetAccessLog() function is called to acquire authentication record stored in the terminal.
To acquire log data from the terminal, UCSAPI_GetAccessLog() function must enter log type suitable to receive them on the third parameter. Allowed log type for input are UCSAPI_TYPE_ALL_NEW, UCSAPI_TYPE_ALL_OLD and UCSAPI_TYPE_ALL_ALL.
UCSAPI_TYPE_ALL_NEW is a type to acquire log that has not been transmitted to the server, and UCSAPI_TYPE_ALL_OLD is a type to acquire log that has already been transmitted to the server. Use UCSAPI_TYPE_ALL_ALL type to receive all these two types.

```
UCSAPI_RETURN ret = UCSAPI_GetAccessLog(m_ClientID, TerminalID, UCSAPI_TYPE_ALL_NEW);

if(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail

// Callback event on UCSAPI_GetAccessLog
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
```

```
        pi = (UCSAPI_PACKET_INFO*)wParam;
        UCSAPI_LOG_DATA LogData;
        switch(EventType)
        {
        case UCSAPI_EVENT_GETACCESSLOG:
            if(pi.ErrorCode == 0)
                // Success
                memcpy(&LogData, (UCSAPI_LOG_DATA*)lParam, sizeof(UCSAPI_LOG_DATA));
            else
                // Fail
            break;
        }
    }
```

Acquiring the Number of Log Data

UCSAPI_GetAccessLogCount() function is called to acquire the number of authentication records stored in the terminal.

```
    UCSAPI_RETURN ret = UCSAPI_GetAccessLogCount(m_ClientID, TerminalID,
UCSAPI_TYPE_ALL_NEW);

    if(ret == UCSAPIERR_NONE)
        //Success
    else
        //Fail

    // Callback event on UCSAPI_GetAccessLogCount
    LRESULT CALLBACK UCSAPINotifyCallback(
            UINT TerminalID,
            UINT EventType,
            WPARAM wParam,
            LPARAM lParam)
    {
            UCSAPI_PACKET_INFO* pi;
            pi = (UCSAPI_PACKET_INFO*)wParam;
            uint32_t LogCount;
            switch(EventType)
```

```
        {
        case UCSAPI_EVENT_GETACCESSLOG:
            if(pi.ErrorCode == 0)
                // Success
                LogCount = (uint32_t)lParam;
            else
                // Fail
            break;
        }
}
```

7.1.4 Authenticating Server

UCSAPI module provides API to respond to terminal's request.
The terminal creates UCSAPI_EVENT_REQUEST event to request work from the server, and the server must transmits the result to the server after processing request.
These works usually occur while the terminal is working in server authentication mode (N/S or NO), and the application program must store user's authentication information in local database to perform server authenticating function.
Also, to perform fingerprint authentication, UCB (UNION COMMUNITY Biometric) SDK that provides fingerprint registration and authentication function needs to be used.

Responding to User Authentication Type Request

To perform 1:1 authentication procedure, user's authentication type information is required and the terminal creates UCSAPI_EVENT_REQUEST_VERIFYINFO event to the server to acquire it. The application program transmits requested user's event type and authentication authority information to the terminal. If authentication authority information acquired from the server has authentication capable value (TRUE), the terminal collects user's fingerprint sample (or password) and makes authentication request to the server. Otherwise, authentication procedure is considered as authentication failure and is terminated.

```
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO pi;
        UCSAPI_VERIFY_INFO vi;


        switch(EventType)
        {
        case UCSAPI_EVENT_REQUEST_VERIFYINFO:
             memcpy(&vi, (UCSAPI_VERIFY_INFO*)lParam, sizeof(UCSAPI_VERIFY_INFO));
             PacketInfo.ClientID = 0;
             PacketInfo.ErrorCode = 0;

             vi.AuthType = UCSAPI_AUTHTYPE_FP;
             vi.IsAccessibility = TRUE;
```

```
            vi.UserID = SearchedUserID;

            UCSAPI_ServerResponse(
                TerminalID,

                UCSAPI_EVENT_REQUEST_VERIFYINFO,
                (WPARAM)&PacketInfo,
                (LPARAM)&VerifyInfo);
            break;
        }
}
```

Responding to Card Authentication Request

To perform card authentication, the terminal along with entered card number creates UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH event to the server. After comparing card number acquired from the terminal with stored user's card number, the application program transmits the result to the terminal.

```
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO pi;
        UCSAPI_VERIFY_INFO vi;
        CHAR szCardNumer[21] = {0,}
        switch(EventType)
        {
        case UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH:
            memcpy(&vi, (UCSAPI_VERIFY_INFO*)lParam, sizeof(UCSAPI_VERIFY_INFO));
            PacketInfo.ClientID = 0;
            PacketInfo.ErrorCode = 0;
            memcpy(szCardNumber, vi.VerifyData.Data, vi.VerifyData.Length);

            vi.AuthType = UCSAPI_AUTHTYPE_CARD;

            If(IsAccessiility())
            {
                vi.IsAccessibility = TRUE;

                If(strcmp(szCardNumber, ReferenceCardNumber) == 0)
```

```
                {
                    vi.IsMatched = TRUE;
                    vi.ErrorCode = 0;
                }
                else
                {
                    vi.IsMatched = FALSE;
                    vi.ErrorCode = UCSAPIERR_MATCHING;
                }
                vi.UserID = MatchedUserID;
            }
            else
            {
                vi.IsAccessibility = FALSE;
                vi.IsMatched = FALSE;
                vi.ErrorCode = UCSAPIERR_ACCESSIBILITY;
            }

            UCSAPI_ServerResponse(
                    TerminalID,

                    UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH,
                    (WPARAM)&PacketInfo,
                    (LPARAM)&VerifyInfo);
            break;
        }
}
```

Responding to 1:1 Authentication Request

To perform 1:1 authentication, the terminal along with entered fingerprint sample (or password) creates UCSAPI_EVENT_REQUEST_VERIFYMATCH to the server. After comparing fingerprint sample acquired from the terminal with stored user's fingerprint template, the application program transmits the result to the terminal.

```
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO pi;
        UCSAPI_VERIFY_INFO vi;
```

```
        CHAR szCardNumer[21] = {0,}
        switch(EventType)
        {
        case UCSAPI_EVENT_REQUEST_VERIFYMATCH:
            memcpy(&vi, (UCSAPI_VERIFY_INFO*)lParam, sizeof(UCSAPI_VERIFY_INFO));
            PacketInfo.ClientID = 0;
            PacketInfo.ErrorCode = 0;
            memcpy(szCardNumber, vi.VerifyData.Data, vi.VerifyData.Length);

            If(IsAccessibility())
            {
                vi.IsAccessibility = TRUE;

                // Authentication procedure starts
                // ret = UCBioAPI_Verify();
                // Authentication procedure completed

                If(ret == UCBioAPIERROR_NONE)
                {
                    vi.IsMatched = TRUE;
                    vi.ErrorCode = 0;
                    vi.UserID = MatchedUserID;
                }
                else
                {
                    vi.IsMatched = FALSE;
                    vi.ErrorCode = UCSAPIERR_MATCHING;
                }
            }
            else
            {
                vi.IsAccessibility = FALSE;
                vi.IsMatched = FALSE;
                vi.ErrorCode = UCSAPIERR_ACCESSIBILITY;
            }

            UCSAPI_ServerResponse(
                    TerminalID,

                    UCSAPI_EVENT_REQUEST_VERIFYMATCH,
                    (WPARAM)&PacketInfo,
                    (LPARAM)&VerifyInfo);
            break;
        }
}
```

Responding to 1:N Fingerprint Authentication Request

To perform 1:N fingerprint authentication, the terminal along with entered fingerprint makes authentication request to the server and the server must reply with the result to the terminal after comparing fingerprint information acquired from the terminal with users' fingerprint information.

To perform 1:N fingerprint authentication, the terminal along with entered fingerprint sample creates UCSAPI_EVENT_REQUEST_IDENTIFYMATCH event to the server. After comparing fingerprint sample acquired from the terminal with stored user's fingerprint template, the application program transmits the result to the terminal.

```
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO pi;
        UCSAPI_VERIFY_INFO vi;
        CHAR szCardNumer[21] = {0,}
        switch(EventType)
        {
        case UCSAPI_EVENT_REQUEST_VERIFYMATCH:
            memcpy(&vi, (UCSAPI_VERIFY_INFO*)lParam, sizeof(UCSAPI_VERIFY_INFO));
            PacketInfo.ClientID = 0;
            PacketInfo.ErrorCode = 0;
            memcpy(szCardNumber, vi.VerifyData.Data, vi.VerifyData.Length);

            If(IsAccessibility())
            {
                vi.IsAccessibility = TRUE;

                // Authentication procedure starts
                // for(int I = 0; I < UserCount; I ++)
                // {
                //      ret = UCBioAPI_Verify();
                //      If(ret == UCBioAPIERROR_NONE)
                //      {
                //          memcpy(&UserID, Payload.Data, Payload.Length);
                //          break;
                //      }
                // Authentication procedure completed
```

```
                If(ret == UCBioAPIERROR_NONE)
                {
                    vi.UserID = MatchedUserID;
                    vi.IsMatched = TRUE;
                    vi.ErrorCode = 0;
                }
                else
                {
                    vi.IsMatched = FALSE;
                    vi.ErrorCode = UCSAPIERR_MATCHING;
                }
            }
            else
            {
                vi.IsAccessibility = FALSE;
                vi.IsMatched = FALSE;
                vi.ErrorCode = UCSAPIERR_ACCESSIBILITY;
            }

            UCSAPI_ServerResponse(
                    TerminalID,

                    UCSAPI_EVENT_REQUEST_VERIFYMATCH,
                    (WPARAM)&PacketInfo,
                    (LPARAM)&VerifyInfo);
            break;
        }
}
```

7.1.5 Managing Terminal

Acquiring the Number of Terminals Connected to the Server

To acquired the number of terminals connected to the server, the application program calls UCSAPI_GetTerminalCount() function.

```
uint32_t ConnectionsOfTerminal=0;
UCSAPI_RETURN ret = UCSAPI_GetTerminalCount(&ConnectionsOfTerminal);

if(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail
```

Acquiring Terminal Firmware Version

To acquire terminal's firmware version, the application program calls UCSAPI_GetFirmareVersion() function.

```
UCSAPI_RETURN ret = UCSAPI_GetFirmwareVersion(m_ClientID, TerminalID);

if(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail

// Callback event on UCSAPI_GetFirmwareVersion
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
```

```
        UCSAPI_DATA* VerData;
        char szVersion[30]={0,};
        switch(EventType)
        {
        case UCSAPI_EVENT_GETACCESSLOG:
            if(pi.ErrorCode == 0)
                // Success
                VerData = (UCSAPI_DATA*)lParam;
                memcpy(szVersion, VerData.Data, VerData.Length);
            else
                // Fail
            break;
        }
}
```

Upgrading Terminal Firmware

To upgrade terminal's firmware, the application program calls UCSAPI_UpgradeFirmare() function.

```
char szPathOfFirmware[MAX_PATH+1] = {0,};
// Firmware path is absolute path.
// Here, szPathOfFirmware value is filled.
UCSAPI_RETURN ret = UCSAPI_UpgradeFirmware (m_ClientID, TerminalID, szPathOfFirmware);

if(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail

// Callback event on UCSAPI_UpgradeFirmware
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
```

```
        UCSAPI_PROCESSING_INFO Processing;
        switch(EventType)
        {
        case UCSAPI_EVENT_FW_UPGRADING
            if(pi.ErrorCode == 0)
                memcpy(&Processing,                    (UCSAPI_PROCESSING_INFO*)lParam,
sizeof(UCSAPI_PROCESSING_INFO));
            break;
        case UCSAPI_EVENT_FW_UPGRADE:
            if(pi.ErrorCode == 0)
                // Success
            else
                // Fail
            break;
        }
  }
```

Acquiring Terminal Basic Option Value

To acquire terminal's basic option value, the application program calls UCSAPI_GetBasicTerminalOption() function.

```
  UCSAPI_RETURN ret = UCSAPI_GetBasicTerminalOption(m_ClientID, TerminalID);

  if(ret == UCSAPIERR_NONE)
      //Success
  else
      //Fail

  // Callback event on UCSAPI_GetBasicTerminalOption
  LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
  {
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        UCSAPI_BASIC_TERMINAL_OPTIONS BasicOptions;
        switch(EventType)
```

```
        {
        case UCSAPI_EVENT_GET_BASIC_TERMINAL_OPTION
            if(pi.ErrorCode == 0)
                memcpy(&BasicOptions,(UCSAPI_BASIC_TERMINAL_OPTIONS*)lParam,
                        sizeof(UCSAPI_BASIC_TERMINAL_OPTIONS));
            break;
        }
}
```

Acquiring Terminal Expanded Option Value

To acquire terminal's expanded option value, the application program calls UCSAPI_GetExpandTerminalOption() function.

```
UCSAPI_RETURN ret = UCSAPI_GetExpandTerminalOption(m_ClientID, TerminalID);

if(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail

// Callback event on UCSAPI_GetExpandTerminalOption
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        UCSAPI_EXPAND_TERMINAL_OPTIONS ExpandOptions;
        switch(EventType)
        {
        case UCSAPI_EVENT_GET_EXPAND_TERMINAL_OPTION
            if(pi.ErrorCode == 0)
                memcpy(&ExpandOptions,(UCSAPI_EXPAND_TERMINAL_OPTIONS*)lParam,
                        sizeof(UCSAPI_EXPAND_TERMINAL_OPTIONS));
            break;
        }
```

```
}
```

Setting Terminal Option Value

To set terminal's option value, the application program calls UCSAPI_SetTerminalOption() function.

```
uint8_t BasicOptionFlag = 0, ExpandOptionFlag = 0;


//Option flag is set with combination of items on options allowed for setting.
BasicOptionFlag |= UCSAPI_OPTION_IDLENGTH | UCSAPI_OPTION_SCHEDULE;
ExpandOptionFlag |= UCSAPI_OPTION_OPERATIONMODE | UCSAPI_OPTION_NAME;

UCSAPI_BASIC_TERMINAL_OPTIONS BasicOption;
memset(&BasicOption, 0, sizeof UCSAPI_BASIC_TERMINAL_OPTIONS);
UCSAPI_EXPAND_TERMINAL_OPTIONS ExpandOption;
memset(&ExpandOption, 0, sizeof UCSAPI_EXPAND_TERMINAL_OPTIONS);
BasicOption.IdLength = 4;
Schedule.ScheduleType = 1;
//The below scheduling is a setting example to lock the terminal from 9:30 to 18:30.
Schedule.ScheduleData.Expand.THU.Lock1.IsUsed = 1;
Schedule.ScheduleData.Expand.THU.Lock1.StartHour = 9;
Schedule.ScheduleData.Expand.THU.Lock1.StartMin = 30;
Schedule.ScheduleData.Expand.THU.Lock1.EndHour = 18;
Schedule.ScheduleData.Expand.THU.Lock1.EndMin = 30;
ExpandOption.OperationMode = 1;
sprintf((char*)ExpandOption.PrintName, "%s", "UNION");
ret = UCSAPI_SetTerminalOption(
        m_ClientID,
        TerminalID,
        BasicOptionFlag,
        ExpandOptionFlag,
        &BasicOption,
        &ExpandOption,
        &Schedule);
if(ret == UCSAPIERR_NONE)
    //Success
else
    //Fail
```

```
// Callback event on UCSAPI_SetTerminalOption
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        UCSAPI_EXPAND_TERMINAL_OPTIONS ExpandOptions;
        switch(EventType)
        {
        case UCSAPI_EVENT_SET_TERMINAL_OPTION
            if(pi.ErrorCode == 0)
                //Success
            break;
        }
}
```

Forcefully Opening Terminal Lock Device

To forcefully open the lock device attached to the terminal, the application program calls UCSAPI_OpenTerminal() function.

```
UCSAPI_RETURN ret = UCSAPI_OpenTerminal(m_ClientID, TerminalID);

if(ret == UCSAPIERR_NONE)
      //Success
else
      //Fail

// Callback event on UCSAPI_OpenTerminal
LRESULT CALLBACK UCSAPINotifyCallback(
        UINT TerminalID,
        UINT EventType,
        WPARAM wParam,
        LPARAM lParam)
{
```

```
        UCSAPI_PACKET_INFO* pi;
        pi = (UCSAPI_PACKET_INFO*)wParam;
        switch(EventType)
        {
        case UCSAPI_EVENT_TERMINAL_OPEN
            if(pi.ErrorCode == 0)
                // Success
            break;
        }
}
```

7.2 UCSCOM Module Programming

UCSCOM module was developed to support RAD tool (Visual Basic, Delphi and etc.) developers and web developers (IIS) but not all functions supported in UCSAPI module are supported.
UCSCOM module's method and property are explained in detail in chapter 6 UCBCOM Methods and Properties.

7.2.1 Visual Basic Programming

1. Constructing COM Object

To use COM module, the application program must construct object on the module first.
To refer to COM module in the application program, include UCBCOM 1.0 type library in usable reference item in Reference of project menu of Visual Basic.

COM Object Construction

```
Dim UCSCOMObj As UCSCOMLib.UCSCOMObj      'Declaration UCSCOM Object
Set UCSCOMObj = New UCSCOMLib.UCSCOMObj 'Object construction
```

COM Object Destruction

```
Set UCsCOMObj = nothing
```

2. Starting and Terminating Server

To start server function, the application program must call ServerStart() method.

ServerStart() method puts terminal's connection to the designated connection port always on standby. Also, the application program must call ServerStop() method to terminate server function.

Starting Server Initialization

```
Dim MaxClient As Long
Dim Port As Long
MaxClient = 1
Port = 2201

UCSCOMObj.ServerStart(MaxClient, 2201)
If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    ' Success
Else
    ' Failed
End If

//Related event 1
Private Sub UCSCOMObj_EventTerminaConnected(ByVal TerminalID As Long, ByVal TerminalIP As String)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event 2
Private Sub UCSCOMObj_EventTerminaDisconnected(ByVal TerminalID As Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If
```

Terminating Server

```
UCSCOMObj.ServerStop()
If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    ' Success
Else
    ' Failed
End If
```

3. Managing Terminal User

UCSCOM module provides function to add or delete terminal's users by the application program.

Adding User

To add user information to the terminal, the application program calls AddUser() method.

```
Dim AuthType As Long
Dim IsAdmin As Long
Dim DateLimit As String
Dim TimeLimit As String
Dim SecuLevel As Long

AuthType = UCSAPI_AUTHTYPE_FP
IsAdmin = 0
DateLimit = "0000000000000000" 'YYYYMMDDYYYYMMDD
TimeLimit = "00000000"          'HHmmHHmm
SecuLevel = 0                   'In case of using terminal's authentication level, 0 is designated.
                                    'In case authentication level is 0, terminal uses terminal's
                                     'authentication level.
                                    'To separately designate user's authentication level, value
                                        'other than 0 must be used.

'It is used when authentication type is combination of UCSAPI_AUTHTYPE_PW.
UCSCOMObj.AddMethodPassword(strPassword);
'It is used when authentication type is combination of UCSAPI_AUTHTYPE_CARD.
UCSCOMObj.AddMethodRFID(strRFID);
'It is used when authentication type is combination of UCSAPI_AUTHTYPE_FP.
UCSCOMObj.AddMethodFinger(bInitialize, nSrcFPDataType, nFPDataSize, FPData1, FPData2);


UCSCOMObj.AddUser(
    ClientID,
    TerminalID,
    UserID,
    UserName,
    IsAdmin,
    DateLimit, ' In case of no authentication restriction on period, NULL can be entered.
    TimeLimit, ' In case of no authentication restriction on time, NULL can be entered.
    SecuLevel,
    AuthType)


If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
```

```
    ' Success
Else
    ' Failed

//Related event
Private Sub UCSCOMObj_EventAddUser(ByVal ClientID As Long, ByVal TerminalID As Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If
End Sub
```

Deleting User

To delete terminal user from the terminal, the application program calls DeleteUser() method.

To delete all users in the terminal, set UserID to –1.

```
UCSCOMObj.DeleteUser(ClientID, TerminalID, UserID)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event
Private Sub UCSCOMObj_EventDeleteUser(ByVal ClientID As Long, ByVal TerminalID As Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

End Sub
```

Acquiring the Number of Users

To acquire the number of administrators and general users registered in the terminal, the application program calls GetUserCount() method.

```
UCSCOMObj. GetUserCount(ClientID, TerminalID)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event
Private Sub UCSCOMObj_EventGetUserCount(ByVal ClientID As Long, ByVal TerminalID As Long,
                                        ByVal AdminCount As Long, ByVal UserCount As Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

End Sub
```

Acquiring User List

To acquire ID list information of users registered in the terminal, the application program calls GetUserList() method.

```
UCSCOMObj. GetUserList(ClientID, TerminalID)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If
```

```
//Related event
Private Sub UCSCOMObj_EventGetUserList(ByVal ClientID As Long, ByVal TerminalID As Long,
                                       ByVal UserID As Long, ByVal Admin As Long, ByVal
                                       CurrentBlock As Long, ByVal TotalBlock As Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Fail
End If

End Sub
```

Acquiring User Data

To acquire user information registered in the terminal, the application program calls GetUserData() method.

```
UCSCOMObj. GetUserData(ClientID, TerminalID, UserID)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event
Private Sub UCSCOMObj_EventGetUserData(ByVal ClientID As Long, ByVal TerminalID As Long,
                                       ByVal UserID As Long, ByVal UserName As String, ByVal
                                       Admin As Long, ByVal DateLimit As String, ByVal
                                       TimeLimit As String, ByVal SecuLevel As Long, ByVal
                                       AuthType As Long, ByVal TextPW As String, ByVal
                                       TextRFID As String)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If
```

```
End Sub
```

4. Managing Terminal Log

The terminal has different log storing method according to working mode.

First, authentication log must be stored in the application program when the terminal works in N/S mode. Only if the application program fails to store authentication log, it is stored in the terminal. This method depends on the application program for most of authentication work, log recovery becomes difficult when application program's data are damaged.

On the other hand, authentication log is stored when the terminal works in S/N mode and simultaneously transmitted to the application program. This method allows authentication log stored in two places and there reduces danger of losing log.

Acquiring Log Data

GetAccessLog() function is called to acquire authentication record stored in the terminal.

To acquire log data from the terminal, GetAccessLog() function must enter log type suitable to receive them on the third parameter. Allowed log type for input are UCSAPI_TYPE_ALL_NEW, UCSAPI_TYPE_ALL_OLD and UCSAPI_TYPE_ALL_ALL.

UCSAPI_TYPE_ALL_NEW is a type to acquire log that has not been transmitted to the server, and UCSAPI_TYPE_ALL_OLD is a type to acquire log that has already been transmitted to the server. Use UCSAPI_TYPE_ALL_ALL type to receive all these two types.

```
UCSCOMObj. GetAccessLog(ClientID, TerminalID, UCSAPI_TYPE_ALL_NEW)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event
Private Sub UCSCOMObj_EventGetAccessLog(ByVal Handle As Long, ByVal TerminalID As Long,
                                        ByVal UserID As Long, ByVal AccessTime As String, ByVal
                                        AuthMode As Long, ByVal AuthType As Long, ByVal Result
                                        As Long, ByVal CurrentBlock As Long, ByVal TotalBlock As
                                        Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
```

```
Else
    'Failed
End If

End Sub
```

Acquiring the Number of Log Data

To acquire the number of authentication records stored in the terminal, GetAccessLogCount() method is called.

```
UCSCOMObj. GetAccessLogCount(ClientID, TerminalID, UCSAPI_TYPE_ALL_NEW)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event
Private Sub UCSCOMObj_EventGetAccessLogCount(ByVal Handle As Long, ByVal TerminalID As
                                        Long, ByVal LogCount As Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

End Sub
```

5. Authenticating Server

The terminal makes work request to the server and the server must transmit the result to the terminal after processing requested work.

These works usually occur while the terminal is working in server authentication mode (N/S or NO), and the server must store user information for authentication in local database to perform authentication.

Also, UCB (UNION COMMUNITY Biometric) SDK providing fingerprint registration and authentication function to perform fingerprint authentication.

Responding to User Authentication Type Request

To perform 1:1 authentication procedure, user's authentication type information is required and the terminal creates UCSAPI_EVENT_REQUEST_VERIFYINFO event to the server to acquire it. The application program transmits requested user's event type and authentication authority information to the terminal. If authentication authority information acquired from the server has authentication capable value (TRUE), the terminal collects user's fingerprint sample (or password) and makes authentication request to the server. Otherwise, authentication procedure is considered as authentication failure and is terminated.

```
  Private Sub UCSCOMObj_EventRequestVerifyInfo(ByVal TerminalID As Long, ByVal UserID As
Long)

  Dim AuthType As Long
  Dim IsAccessibility As Long
  Dim ErrorCode As Long

  AuthType = UCSAPI_AUTHTYPE_FP ' In case user's authentication type is fingerprint,
  IsAccessibility = 1 'Authentication authority exists. In case authentication authority does not 'exist,
                        this value is 0.
  ErrorCode = 0 ' No error
  ResponseVerifyInfo(TerminalID, UserID, AuthType, IsAccessibility, ErrorCode)


  If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
      'Success
  Else
      'Failed
  End If

  End Sub
```

Responding to Card Authentication Request

To perform card authentication, the terminal along with entered card number creates UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH event to the server. After comparing card number acquired from the terminal with stored user's card number, the application program transmits the result to the terminal.

```
Private Sub UCSCOMObj_EventRequestCardVerifyMatch(ByVal TerminalID As Long, ByVal AuthMode As Long, ByVal TextRFID As String)

Dim AuthType As Long
Dim IsAccessibility As Long
Dim IsMatched As Long
Dim ErrorCode As Long
Dim MatchedUserID As Long

AuthType = UCSAPI_AUTHTYPE_FP ' In case user's authentication type is fingerprint,
IsAccessibility = 1 'Authentication authority exists. In case authenticating authority does not 'exist,
                    this value is 0.
IsMatched = 1 'In case authentication performance result is successful,
ErrorCode = 0 'No error
ResponseCardVerifyMatch(TerminalID, MatchedUserID, IsAccessibility, IsMatched, ErrorCode)


If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

End Sub
```

Responding to 1:1 Authentication Request

To perform 1:1 authentication, the terminal along with entered fingerprint sample (or password) creates UCSAPI_EVENT_REQUEST_VERIFYMATCH to the server. After comparing fingerprint sample acquired from the terminal with stored user's fingerprint template, the application program transmits the result to the terminal.

```
Private Sub UCSCOMObj_EventRequestVerifyMatch(ByVal TerminalID As Long, ByVal UserID As Long, ByVal AuthMode As Long, ByVal SecuLevel As Long, ByVal AuthType As Long, ByVal VerifyData As String)
```

```
  Dim AuthType As Long
  Dim IsAccessibility As Long
  Dim IsMatched As Long
  Dim ErrorCode As Long
  Dim MatchedUserID As Long

  AuthType = UCSAPI_AUTHTYPE_FP 'In case user's authentication type is fingerprint,
  IsAccessibility = 1 'Authentication authority exists. In case authentication authority does not 'exist,
                        this value is 0.
  IsMatched = 1 'In case authentication performance result is successful,
  ErrorCode = 0 'No error
  ResponseVerifyMatch(TerminalID, MatchedUserID, IsAccessibility, IsMatched, ErrorCode)


  If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
      'Success
  Else
      'Failed
  End If

End Sub
```

Responding to 1:N Fingerprint Authentication Request

To perform 1:N fingerprint authentication, the terminal along with entered fingerprint makes authentication request to the server and the server must reply with the result to the terminal after comparing fingerprint information acquired from the terminal with users' fingerprint information.

To perform 1:N fingerprint authentication, the terminal along with entered fingerprint sample creates UCSAPI_EVENT_REQUEST_IDENTIFYMATCH event to the server. After comparing fingerprint sample acquired from the terminal with stored user's fingerprint template, the application program transmits the result to the terminal.

```
  Private Sub UCSCOMObj_EventRequestIdentifyMatch(ByVal TerminalID As Long, ByVal AuthMode
  As Long, ByVal InputUserIDLength As Long, ByVal SecuLevel As Long, ByVal AuthType As Long,
  ByVal VerifyData As String)

  Dim AuthType As Long
  Dim IsAccessibility As Long
  Dim IsMatched As Long
  Dim ErrorCode As Long

  AuthType = UCSAPI_AUTHTYPE_FP 'In case user's authentication type is fingerprint,
  IsAccessibility = 1 'Authentication authority exists. In case authentication authority does not 'exist,
```

```
                    this value is 0.
  IsMatched = 1 'In case authentication performance result is successful,
  ErrorCode = 0 'No error

  'UserID is a user ID with successful authentication.
  ResponseIdentifyMatch(TerminalID, UserID, IsAccessibility, IsMatched, ErrorCode)


  If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
      'Success
  Else
      'Failed
  End If

End Sub
```

6. Managing Terminal

Acquiring the Number of Terminals Connected to the Server

To acquire the number of terminals connected to the server, the application program calls GetTerminalCount() method.

```
UCSCOMObj.GetTerminalCount()

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event
//None
```

Acquiring Terminal Firmware Version

To acquire terminal's firmware version, the application program calls GetFirmareVersion() method.

```
UCSCOMObj.GetFirmware(ClientID, TerminalID)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

//Related event
Private Sub UCSCOMObj_EventFirmwareVersion(ByVal ClientID As Long, ByVal TerminalID As
                                    Long, ByVal Version As String)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
```

```
Else
    'Failed
End If

End Sub
```

Upgrading Terminal Firmware

To upgrade terminal's firmware, the application program calls UpgradeFirmare() function.

```
UCSCOMObj.GetFirmware(ClientID, TerminalID)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

'Related event 1 (event under upgrading)
Private Sub UCSCOMObj_EventFirmwareUpgrading(ByVal ClientID As Long, ByVal TerminalID As
                                    Long, ByVal CurrentBlock As Long, ByVal TotalBlock As
                                    Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

End Sub

'Related event 2 (upgrade completed event)
Private Sub UCSCOMObj_EventFirmwareUpgrade(ByVal ClientID As Long, ByVal TerminalID As
                                    Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
```

```
End If
End Sub
```

Acquiring and Setting Terminal Option Value

UCSCOM module does not support function to manage terminal option.

Forcefully Opening Terminal Lock Device

To forcefully open the lock device attached to the terminal, the application program calls OpenTerminal() method.

```
UCSCOMObj.OpenTerminal(ClientID, TerminalID)

If UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

'Related event
Private Sub UCSCOMObj_EventOpenTerminal(ByVal ClientID As Long, ByVal TerminalID As Long)
If UCSCOMObj.EventError = UCSAPIERR_NONE Then
    'Success
Else
    'Failed
End If

End Sub
```

7.2.2 Delphi Programming

1. Constructing COM Object

To use COM module, object on the module must be constructed first.

COM Object Construction

```
UCSCOMObj : Variant;
UCSCOMObj := CreateOleObject('UCSCOMLib.UCSCOMObj');
```

COM Object Destruction

```
//Delete object after use.
UCSCOMObj := null;
Set UCSCOMObj = nothing;
```

2. Starting and Terminating Server

To start server function, the application program must call ServerStart() method.

ServerStart() method puts terminal's connection to the designated connection port always on standby. Also, the application program must call ServerStop() method to terminate server function.

Starting Server Initialization

```
Var MaxClient: Integer;
Var Port: Integer;

MaxClient = 1;
Port = 2201;

UCSCOMObj.ServerStart(MaxClient, 2201);
if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;

//Related event
procedure EventTerminalConnected(TerminalID: Integer, TerminalIP: WideString);
procedure EventTerminalDisconnected(TerminalID: Integer);
```

Terminating Server

```
UCSCOMObj.ServerStop();

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;
```

3. Managing Terminal User

UCSCOM module provides function to add or delete terminal's user by the application program.

Adding User

To add user information to the terminal, the application program calls AddUser() method.

```
Var AuthType : Long;
Var IsAdmin : Long;
Var DateLimit : String;
Var TimeLimit : String;
Var SecuLevel : Long;

AuthType = UCSAPI_AUTHTYPE_FP;
IsAdmin = 0;
DateLimit = "0000000000000000"; //YYYYMMDDYYYYMMDD
TimeLimit = "00000000";          //HHmmHHmm
SecuLevel = 0;                   //In case terminal's authentication level is used, 0 is designated.
                                 //In case authentication level is 0, terminal uses terminal's
                                 //authentication level value.
                                 //To separately designate user's authentication level, value
                                 //other than 0 must be used.

//It is used when authentication type is combination of UCSAPI_AUTHTYPE_PW.
UCSCOMObj.AddMethodPassword(strPassword);
//It is used when authentication type is combination of UCSAPI_AUTHTYPE_CARD.
UCSCOMObj.AddMethodRFID(strRFID);
//It is used when authentication type is combination of UCSAPI_AUTHTYPE_FP.
UCSCOMObj.AddMethodFinger(bInitialize, nSrcFPDataType, nFPDataSize, FPData1, FPData2);


UCSCOMObj.AddUser(
    ClientID,
    TerminalID,
    UserID,
    UserName,
    IsAdmin,
    DateLimit, // In case of no authentication restriction on period, NULL can be entered.
    TimeLimit, // In case of no authentication restriction on time, NULL can be entered.
    SecuLevel,
    AuthType,
    NULL, //It is used when authentication type is combination of UCSAPI_AUTHTYPE_PW.
    NULL  //It is used when authentication type is combination of UCSAPI_AUTHTYPE_CARD.
    TextBDBTemplate); //It is used when authentication type is combination of
UCSAPI_AUTHTYPE_FP.
```

```
                        //It is fingerprint template of UCSAPI_BDB_FORM_FULLBDB.

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;

//Related event
procedure EventAddUser(ClientID: Integer, TerminalID: Integer);
```

Deleting User

To delete user from the terminal, the application program calls DeleteUser() method.

To delete all users in the terminal, set UserID to -1.

```
UCSCOMObj.DeleteUser(ClientID, TerminalID, UserID);

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;


//Related event
procedure EventDeleteUser(ClientID: Integer, TerminalID: Integer, UserID: Integer);
```

Acquiring the Number of Users

To acquire the number of administrators and general users, the application program calls GetUserCount() method.

```
UCSCOMObj.GetUserCount(ClientID, TerminalID)

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
```

```
        //Success
    end;

//Related event
procedure EventGetUserCount(ClientID: Integer; TerminalID: Integer; AdminCount: Integer;
                                                                    UserCount: Integer);
```

Acquiring User List

To acquire ID list information of users registered in the terminal, the application program calls GetUserList() method.

```
UCSCOMObj.GetUserList(ClientID, TerminalID);

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;


//Related event
procedure EventGetUserList(ClientID: Integer; TerminalID: Integer; UserID: Integer;
                                    Admin: Integer; CurrentBlock: Integer; TotalBlock: Integer);
```

Acquiring User Data

To acquire user information registered in the terminal, the application program calls GetUserData() method.

```
UCSCOMObj.GetUserData(ClientID, TerminalID, UserID);

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
    //Success
    end;
```

```
//Related event
procedure EventGetUserData(ClientID: Integer; TerminalID: Integer; UserID: Integer;
                           const UserName: WideString; Admin: Integer;
                           const DateLimit: WideString; const TimeLimit: WideString;
                           SecuLevel: Integer; AuthType: Integer; const TextPW: WideString;
                           const TextRFID: WideString);
```

4. Managing Terminal Log

The terminal has different log storing method according to working mode.

First, authentication log must be stored in the application program when the terminal works in N/S mode. Only if the application program fails to store authentication log, it is stored in the terminal. This method depends on the application program for most of authentication work, log recovery becomes difficult when application program's data are damaged.

On the other hand, authentication log is stored when the terminal works in S/N mode and simultaneously transmitted to the application program. This method allows authentication log stored in two places and there reduces danger of losing log.

Acquiring Log Data

GetAccessLog() function is called to acquire authentication record stored in the terminal.

To acquire log data from the terminal, GetAccessLog() function must enter log type suitable to receive them on the third parameter. Allowed log type for input are UCSAPI_TYPE_ALL_NEW, UCSAPI_TYPE_ALL_OLD and UCSAPI_TYPE_ALL_ALL.

UCSAPI_TYPE_ALL_NEW is a type to acquire log that has not been transmitted to the server, and UCSAPI_TYPE_ALL_OLD is a type to acquire log that has already been transmitted to the server. Use UCSAPI_TYPE_ALL_ALL type to receive all these two types.

```
UCSCOMObj.GetAccessLog(ClientID, TerminalID, UCSAPI_TYPE_ALL_NEW);

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;


//Related event
procedure EventGetAccessLog(ClientID: Integer; TerminalID: Integer; UserID: Integer;
                            const AccessTime: WideString; AuthMode: Integer; AuthType:
                                Integer;
                            Result: Integer; CurrentBlock: Integer; TotalBlock: Integer);
```

Acquiring the Number of Log Data

To acquired the number of authentication records stored in the terminal, GetAccessLogCount() method is called.

```
UCSCOMObj.GetAccessLogCount(ClientID, TerminalID, UCSAPI_TYPE_ALL_NEW)

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;


//Related event
procedure EventGetAccessLogCount(ClientID: Integer; TerminalID: Integer; LogCount: Integer);
```

5. Authenticating Server

The terminal makes work request to the server and the server must transmit the result to the terminal after processing requested work.

These works usually occur while the terminal is working in server authentication mode (N/S or NO), and the server must store user information for authentication in local database to perform authentication.

Also, UCB (UNION COMMUNITY Biometric) SDK providing fingerprint registration and authentication function to perform fingerprint authentication.

Responding to User Authentication Type Request

To perform 1:1 authentication procedure, user's authentication type information is required and the terminal creates UCSAPI_EVENT_REQUEST_VERIFYINFO event to the server to acquire it. The application program transmits requested user's event type and authentication authority information to the terminal. If authentication authority information acquired from the server has authentication capable value (TRUE), the terminal collects user's fingerprint sample (or password) and makes authentication request to the server. Otherwise, authentication procedure is considered as authentication failure and is terminated.

```
procedure EventRequestVerifyInfo(TerminalID: Integer; UserID: Integer);

// Responding method to event
var
AuthType: Integer;
IsAccessibility: Integer;
ErrorCode: Integer;

AuthType = UCSAPI_AUTHTYPE_FP;   //In case user's authentication type is fingerprint
IsAccessibility = 1; //Authentication authority exists. In case of no authentication authority, //this
                     value becomes 0.
ErrorCode = 0;   //No error
UCSCOMObj.ResponseVerifyInfo(TerminalID, UserID, AuthType, IsAccessibility, ErrorCode);
```

Responding to Card Authentication Request

To perform card authentication, the terminal along with entered card number creates UCSAPI_EVENT_REQUEST_CARDVERIFYMATCH event to the server. After comparing card number acquired from the terminal with stored user's card number, the application program transmits the result to the terminal.

```
procedure EventRequestCardVerifyMatch(TerminalID: Integer; AuthMode: Integer;
                                      const TextRFID: WideString);


// Responding method to event
var
AuthType: Integer;
IsAccessibility: Integer;
IsMatched: Integer;
ErrorCode: Integer;
MatchedUserID: Integer;

AuthType = UCSAPI_AUTHTYPE_FP;   // In case user's authenticating type is fingerprint
IsAccessibility = 1;   //Authentication authority exists. In case of no authentication authority, //this
                       value becomes 0.
IsMatched = 1;   //In case authentication performance result is successful
ErrorCode = 0;   //No error
UCSCOMObj.ResponseCardVerifyMatch(TerminalID, MatchedUserID, IsAccessibility, IsMatched,
ErrorCode);
```

Responding to 1:1 Authentication Request

To perform 1:1 authentication, the terminal along with entered fingerprint sample (or password) creates UCSAPI_EVENT_REQUEST_VERIFYMATCH to the server. After comparing fingerprint sample acquired from the terminal with stored user's fingerprint template, the application program transmits the result to the terminal.

```
procedure EventRequestVerifyMatch(TerminalID: Integer; UserID: Integer; AuthMode: Integer;
                                  SecuLevel: Integer; AuthType: Integer;
                                  const VerifyData: WideString);

//Responding method to event
var
AuthType: Integer;
IsAccessibility: Integer;
IsMatched: Integer;
ErrorCode: Integer;
MatchedUserID: Integer;

AuthType = UCSAPI_AUTHTYPE_FP; //In case user's authentication type is fingerprint
IsAccessibility = 1; //Authentication authority exists. In case of no authentication authority, //this
                     value becomes 1.
IsMatched = 1; //In case authentication performance result is successful
ErrorCode = 0; //No error
```

```
UCSCOMObj.ResponseVerifyMatch(TerminalID, MatchedUserID, IsAccessibility, IsMatched,
ErrorCode);
```

Responding to 1:N Fingerprint Authentication Request

To perform 1:N fingerprint authentication, the terminal along with entered fingerprint makes authentication request to the server and the server must reply with the result to the terminal after comparing fingerprint information acquired from the terminal with users' fingerprint information.

To perform 1:N fingerprint authentication, the terminal along with entered fingerprint sample creates UCSAPI_EVENT_REQUEST_IDENTIFYMATCH event to the server. After comparing fingerprint sample acquired from the terminal with stored user's fingerprint template, the application program transmits the result to the terminal.

```
procedure EventRequestIdentifyMatch(TerminalID: Integer; AuthMode: Integer;
                                    InputUserIDLength: Integer; SecuLevel: Integer;
                                    AuthType: Integer; const VerifyData: WideString);

//Responding method to event
var
AuthType: Integer;
IsAccessibility: Integer;
IsMatched: Integer;
ErrorCode: Integer;
MatchedUserID: Integer;

AuthType = UCSAPI_AUTHTYPE_FP;   //In case user's authentication type is fingerprint
IsAccessibility = 1;   //Authentication authority exists. In case of no authentication authority, //this
                       value becomes 0.
IsMatched = 1;   //In case authentication performance result is successful
ErrorCode = 0;   //No error

//UserID is a user ID with successful authentication.
UCSCOMObj.ResponseIdentifyMatch(TerminalID, MatchedUserID, IsAccessibility, IsMatched,
ErrorCode);
```

6. Managing Terminal

Acquiring the Number of Terminals Connected to the Server

To acquire the number of terminals connected to the server, the application program calls GetTerminalCount() method.

```
UCSCOMObj.GetTerminalCount();

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;

//Related event
//None
```

Acquiring Terminal Firmware Version

To acquire terminal's firmware version, the application program calls GetFirmareVersion() method.

```
UCSCOMObj.GetFirmware(ClientID, TerminalID);

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;


//Related event
procedure EventFirmwareVersion(ClientID: Integer; TerminalID: Integer; const Version:WideString);
```

Upgrading Terminal Firmware

To upgrade terminal's firmware, the application program calls UpgradeFirmare() function.

```
UCSCOMObj.GetFirmware(ClientID, TerminalID);

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;

//Related event 1 (event under upgrading)
procedure EventFirmwareUpgrading(ClientID: Integer; TerminalID: Integer; CurrentBlock: Integer;
                                 TotalBlock: Integer);
//Related event 2 (Upgrade completed event)
procedure EventFirmwareUpgrade(ClientID: Integer; TerminalID: Integer);
```

Acquiring and Setting Terminal Option Value

UCSCOM module does not provide function to manage terminal option.

Forcefully Opening Terminal Lock Device

To open the lock device attached to the terminal, the application program calls OpenTerminal() method.

```
UCSCOMObj.OpenTerminal(ClientID, TerminalID);

if UCSCOMObj.ErrorCode = UCSAPIERR_NONE Then
    begin
        //Success
    end;

//Related event
procedure EventOpenTerminal(ClientID: Integer; TerminalID: Integer);
```

8. Error Handling

Error	Code	Description
UCSAPIERR_NONE	0x000000	No error
UCSAPIERR_NOT_ACTIVE	0x000001	Server is not executing.
UCSAPIERR_MEMORY_ERROR	0x000002	Memory error
UCSAPIERR_INVALID_POINTER	0x000003	I/O function parameter or I/O structure field is not a valid pointer.
UCSAPIERR_INVALID_INPUT_POINTER	0x000004	Input parameter or entered structure is not a valid pointer.
UCSAPIERR_INVALID_DATA	0x000005	Entered parameter is not valid.
UCSAPIERR_INVALID_OUTPUT_POINTER	0x000006	Parameter or structure field for output is not a valid pointer.
UCSAPIERR_OS_ACCESS_DENIED	0x000007	Access to OS resource is restricted.
UCSAPIERR_FUNCTION_FAILED	0x000008	Function failed with unknown reason.
UCSAPIERR_INCOMPATIBLE_VERSION	0x000009	Version information is not compatible.
UCSAPIERR_INVALID_TERMINAL	0x000010	Not a valid terminal ID
UCSAPIERR_START_SERVER	0x000011	Server start failure
UCSAPIERR_FW_FILE_NOTEXIST	0x000012	Firmware can not be found.
UCSAPIERR_FW_DOWNLOAD	0x000013	Firmware upgrade error
UCSAPIERR_LOG_UPLOAD	0x00001	Failure in acquiring authentication record

	4	
UCSAPIERR_TIMEOUT	0x000015	Function failure due to timeout
UCSAPIERR_DATABASE	0x0007D3	Database access error
UCSAPIERR_INVALID_USER	0x000BBC	Unregistered user (It occurs during server authentication.)
UCSAPIERR_MATCHING	0x000BBD	Authentication failure (It occurs during server authentication.)
UCSAPIERR_ACCESSIBILITY	0x000BBE	No authentication authority (It occurs during server authentication.)
UCSAPIERR_ADDUSER	0x001389	Terminal user addition failure
UCSAPIERR_NOT_ENOUGH_MEMORY	0x00138A	Insufficient terminal memory
UCSAPIERR_WRITE_MEMORY	0x00138D	Terminal memory write error (It occurs if another write requests is made while the terminal is writing to flash memory.)
UCSAPIERR_FIND_USER	0x00138E	It occurs when requested user can not be found in the server.)
UCSAPIERR_FIRMWARE_VERSION	0x00138F	It occurs when upgrade can not be accomplished due to incompatible version during firmware upgrading.
UCSAPIERR_UNKNOW_REASON	0x001770	Unknown error
UCSAPIERR_CARD_DUPLICATED	0x001771	Duplicate card number occurred during card registration in the terminal.