UNION COMMUNITY
Server SDK For Windows

Programmer's Guide
Version 4.0

October, 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 where the software is 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 completed, 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 personal 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. However, we will not replace a product damaged due to your mishandling or negligence. UNION COMMUNITY Co., Ltd. does not guarantee that the software and its features will satisfy your specific needs, 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 damaged the program or its components, or failed to comply with the terms described in this agreement.

9. Customer Service

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

10. General Terms

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

11. Contact

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

Table of Contents

1. Overview

UCS (UNION COMMUNITY Server) SDK was created in the form of a high-level SDK to facilitate the development of application programs that can work with the network-type fingerprint recognition device of UNION COMMUNITY.

UCS SDK provides the interface (application Programming Interface, API) required in the development of fingerprint recognition server application programs. It can be used with UCBioBSP SDK for fingerprint registration and authentication.

1.1 Application

UCS SDK defines the server application program interface that can work with network-type terminal products provided by UNION COMMUNITY. It is therefore applied in application areas such as access control, time/attendance, drinking water, and school record management to facilitate easy and stable program development.

1.2 Special Features

■ Centralized Management Type

As a method that connects the terminal to UCS SDK, all terminals can be managed centrally. This type of connection is very useful in configuring a network using public networks.

If public networks using the measured rate system (system that charges according to the duration of line usage) are used, the method that connects directly to the terminal without implementing SDK's server function can be used.

■ Provides various APIs for terminal management

Various APIs are provided for user management, log management and access control management.

■ Provides various development modules and sample sources

UCS SDK provides COM-based modules and .NET class libraries to facilitate development with tools such as Visual Basic, Delphi and DotNET as well as C/C++. Sample sources required in SDK use are also provided.

■ Provides various authentication methods

Authentication methods using tools such as fingerprint, password and card and various combinations of these tools are provided for user identification.

1.3 Development Environment

Modules provided by UCS SDK were compiled at VC++ 6.0, and programming using this SDK can be done using most of 32bit compilers such as Visual C++.

For developers using Visual Basic, Delphi and DotNET as well as C/C++, COM-based modules and .NET class libraries are provided to facilitate development using these tools. Refer to each of sample sources on how to use the modules.

1.4 Module Organization

■ **Basic Module : UCSAPI40.dll**

As a core module of UCS SDK, it is the main module that implements all functions related to communication with the terminal. This module must be included for development using UCS SDK.

APIs that can be used in C/C++ are provided.

Relevant sample codes can be found at the DLL folder of the Samples folder.

■ **COM Module : UCSAPICOM.dll**

It is a COM (Component Object Module) module developed to support users of RAD (Rapid Application Development) tools such as Visual Basic and Delphi.

As UCSAPICOM.dll exists at a higher level compared to UCSAPI40.dll, UCSAPI40.dll is required for operation. Also, this module does not provide all functions provided by UCSAPI40.dll but it has

some functions that are not provided by UCSAPI40.dll.

Relevant sample codes can be found at the COM folder of the Samples folder.

■ **DotNET Module : UNIONCOMM.SDK.UCSAPI40.dll**

This module is a class library for .NET that can be used in languages for the Microsoft .NET environment such as C#, VisualBasic.NET. As in the COM module, UNIONCOMM.SDK.UCSAPI40.dll exists at a higher level compared to UCSAPI40.dll and UCSAPI40.dll is required for operation.

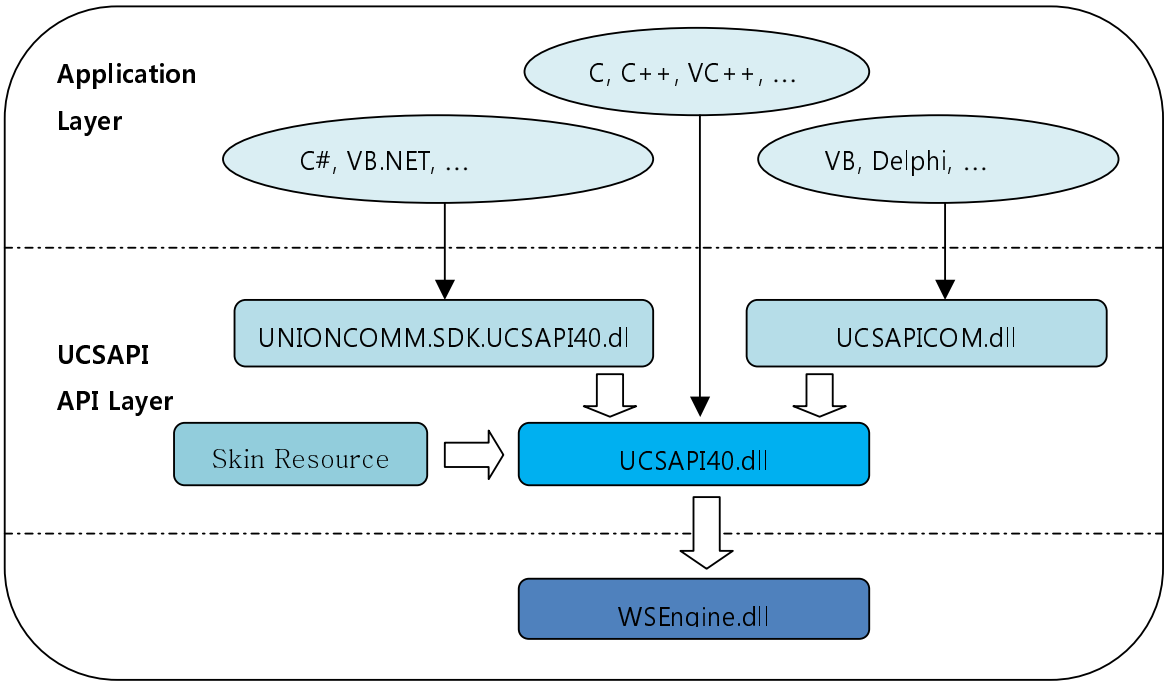
Relevant sample codes can be found at the dotNET folder of the Samples folder.

■ **Winsock Engine Module : WSEngine.dll**

It is the module that handles socket I/O (Input/Output). WSEngine.dll exists at a lower level compared to UCSAPI40.dll

1.5 Development Model

The structure of the development model is shown in the following figure.



1.6 Terminology Description

■ **Terminal**

It is the network-type fingerprint recognition terminal provided by UNION COMMUNITY.

■ **Client**

It is the application program that communicates with the provided network-type fingerprint recognition terminal.

■ **Client ID (ClientID)**

The client ID is used to develop an application program with the client/server model.

The server module requires the key that identifies each client to support the multi-client environment, and this key is called ClientID.

■ **1:1 인증Authentication (1 to 1, Verification)**

It is the 1:1 process that compares the submitted sample with the fingerprint template (or card, password) corresponding to the user ID for user identification.

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

It is the 1:N process that compares the submitted sample with a part of or all fingerprint templates for user identification.

■ **Authentication Type**

The authentication types used during user authentication are defined.

In case the fingerprint recognition terminal uses the server authentication method after the terminal enters UserID or UniqueID for user identification, the terminal requests the authentication type to the server to obtain the authentication type of the user registered in the server.

Type	Value	Contents
1:N Fingerprint	0	1:N fingerprint authentication
1:1 Fingerprint	1	1:1 fingerprint authentication

Card & Fingerprint	2	By storing fingerprints at the smart card, this type implements 1:1 authentication between the input fingerprint and stored fingerprint.
Card	3	Card authentication
Password	4	Password authentication

■ **Registration Authentication Type**

The authentication types that are available during user registration are defined.

Authentication methods using tools for user identification such as fingerprint, password and card and various combinations of these tools are provided.

Type	Contents
Fingerprint	The fingerprint is used as authentication tool.
Fingerprint Card	The Fingerprint card is used as authentication tool. By storing user's fingerprint templates at the smart card, the fingerprint card implements 1:1 authentication between the sample entered during authentication and stored template.
Password	The password is used as authentication tool. The password is a character string value with a maximum of 8 digits.
Card	The card is used as authentication tool.
Card or Fingerprint	The card or fingerprint is used as authentication tool.
Card and Fingerprint	The combination of card and fingerprint is used as authentication tool.
Card or Password	The card or password is used as authentication tool.
Card and Password	The combination of card and fingerprint is used as authentication tool.
(ID and Fingerprint) or (Card and Fingerprint)	The combination of ID and fingerprint or the combination of card and fingerprint is used as authentication tool.
(ID and Password) or (Card and Password)	The combination of ID and password or the combination of card and password is used as authentication tool.
Fingerprint and Password	The combination of fingerprint and password is used as authentication tool.
Fingerprint or Password	After fingerprint authentication fails, the password is used as authentication tool.

Card and Password and Fingerprint	The combination of card, password and fingerprint is used as authentication tool.

■ **Fingerprint Security Level**

The security level to be used during fingerprint authentication is defined. The value ranges from 1~9. UCS SDK recommends developers to use the following level values. If the level value is high, the false reject rate (FRR) is increased and the false acceptance rate (FAR) is decreased. UCS SDK recommends level 4 for 1:1 authentication (verification) and level 5 for 1:N authentication (identification). If FAR is high compared to the recommended default level, an application program can increase the authentication level. On the other hand, if FRR is high, the authentication level can be decreased.

■ **Terminal Authentication Mode**

The operation modes for user's authentication and log record are defined.

Mode	Value	Contents
N / S	0	User authentication is implemented at the server when the network is online and at the terminal when the network is offline. The authentication record is stored at the server during server authentication. If the network is disconnected, the terminal stores the authentication record and sends the stored authentication record to the server when the server is connected.
S / N	1	When the network is online, user authentication is implemented at the terminal. For the user authentication request that is not stored at the terminal, the terminal requests authentication to the server. The authentication record is always stored at the terminal. When the network is online, only the authentication result is sent to the server to be stored.
N / O	2	User authentication is implemented only at the server. If the network is offline, user authentication cannot be implemented.
S / O	3	User authentication is implemented only at the terminal. When the network is online, only the authentication log is sent to the server.
S / S	4	The terminal does not attempt connection to the server but it only waits for the connection of an application program. Authentication is implemented at the terminal.

■ **Terminal Application Mode**

Program operation modes provided by the terminal are defined.

Mode	Value	Contents
Access Control	0	The terminal is operated for access control.
Time/Attendance	1	The terminal is operated for time/attendance management.
Drinking Water	2	The terminal is operated for drinking water management.

■ **User Authentication Mode**

The authentication purposes during user authentication are defined. The expanded authentication mode can be used by setting up the expanded mode option of the terminal. The authentication mode can be used when the program is operated for time/attendance and drinking water management purpose.

Mode	Value	Contents
Office Start	0	Authenticates with office start mode
Office Leave	1	Authenticates with office leave mode
General	2	Authenticates with general mode
Work Outside	3	Authenticates with work outside mode
Return to Office	4	Authenticates with return to office mode

■ **Log Get Type**

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

The number of logs that can be stored internally varies according to the terminal model. If the maximum capacity is exceeded, new logs are stored by deleting some of the existing records.

Type	Value	Contents
New Log	0	New log that was not sent to the server
Old Log	1	Log that was already sent to the server
All Logs	2	All logs stored at the terminal

■ **User Property**

It is the 1byte-sized data field that defines authentication type and indicates whether the user is an administrator or not.

The value of 1 or 0 can be designated to each field. To use the property value of the corresponding

field, the value of 1 is designated.

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

-Admin:

The user can be designated as terminal administrator.

-Identify:

The user can be designated to use 1:N fingerprint authentication.

-Operation:

Each of authentication types can be designated to be used with AND or OR combination.
1 is set for AND combination while 0 is set for OR combination.

-CardID:

RFID can be designated to be used like UserID or UniqueID. CardID does not use card's RFID as authentication tool but instead, the card's RFID is simply used as an identifier like UserID.
It must be designated using AND combination with other authentication type.

-Card:

The user can be designated to use card authentication.

-Password:

The user can be designated to use password authentication.

-Fingerprint:

The user can be designated to use fingerprint authentication.

The user property can be represented 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

⑬ Card and Password and Fingerprint

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

■ **Terminal Admin**

The terminal administrator is a user who has rights to change terminal's setting information and register/delete users. If more than 1 terminal administrator is designated at the terminal, the administrator login process is required when entering the setup screen of the terminal. For the safe use of the terminal, the registration of the terminal administrator is recommended.

■ **Terminal Status**

The terminal periodically or immediately sends to the server the status of the terminal and devices connected to the terminal when the status is changed.

Terminal Lock Status:

This value represents the operability status value of the terminal.

SDK can set up terminal's operation status as lock/unlock. In lock state, the logon and network connection of the terminal are maintained, but the operation and access of the terminal are not allowed. The terminal in lock state does not even allow keypad operation.

Locking Device Control Status:

This value represents the status value of the locking device connected to the terminal.

SDK can set/unset the status of the locking device connected to the terminal as open state. In open state, access is allowed without user authentication.

Locking Device Monitoring Status:

This value represents open/closed status value of the locking device received from the locking device with monitoring support.

Terminal Cover Status:

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

Status	Value	Content
Terminal Status	0	Normal
	1	Lock state
Locking Device Control Status	0	Closed state
	1	Open state
Locking Device Monitoring Status	0	Closed state
	1	Open state
	2	State that is not monitoring
Terminal Cover Open Status	0	Closed state
	1	Open state

< Terminal Status Information Summary >

2. Installation

2.1 System Requirements

■ **CPU**

Intel Pentium 133MHz or higher

■ **Memory**

16MB or higher

■ **USB Port**

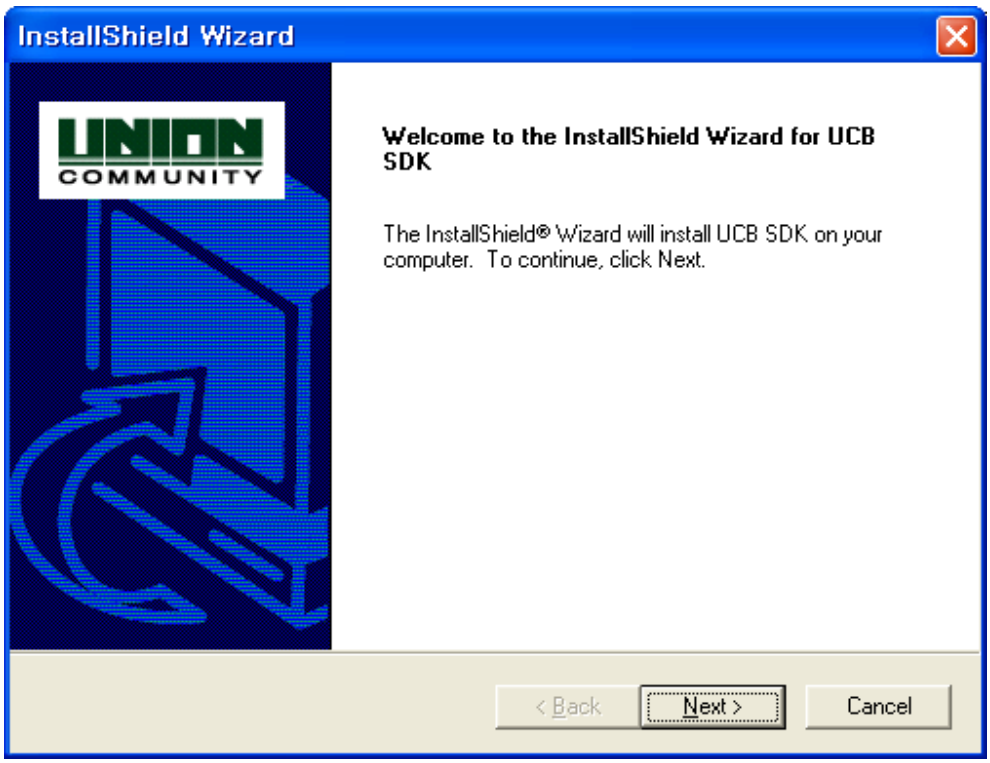
USB 1.1

■ **OS**

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

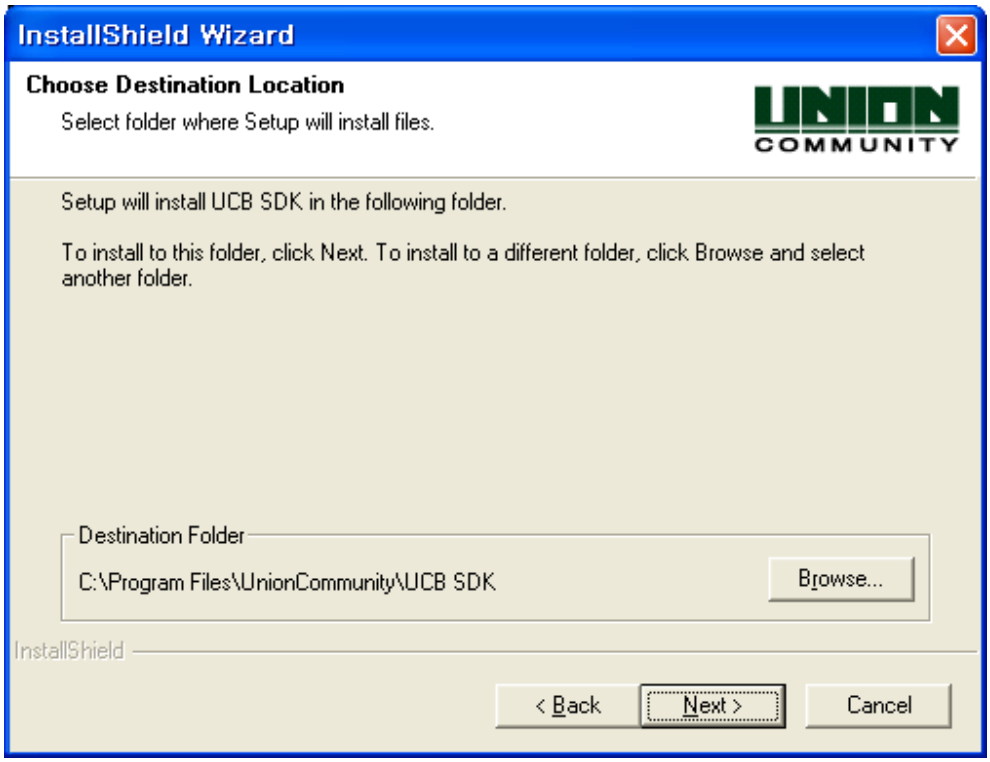
2.2 Installation

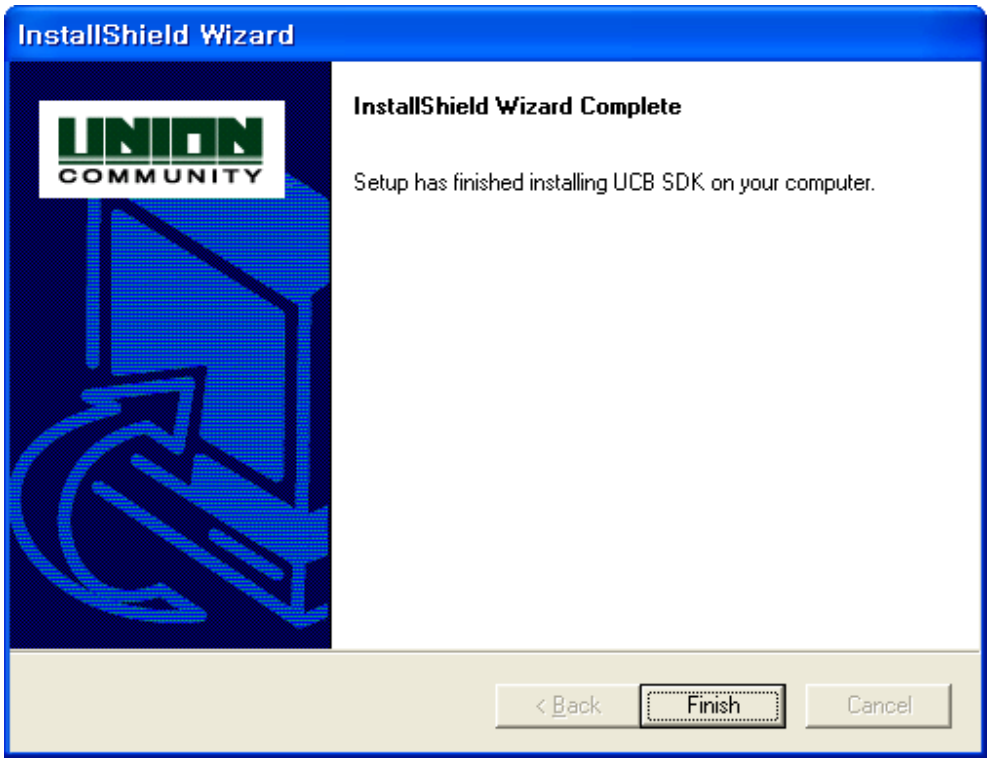
If the installation CD is inserted, Setup.exe is executed automatically.



Follow the step-by-step procedures for installation.

Enter the user information and product serial number.





2.3 Files to be installed

When SDK installation is completed normally, the following files are installed at the designated installation folders.

2.3.1 Windows System Directory

Core modules for the use of SDK are installed. The following files are installed.

UCSAPI40.dll

Core module of UCS SDK. It is in charge of performing all functions of SDK.

UCSAPICOM.dll

COM module for RAD tool developer.

WSEngine.dll

Communication module for Windows socket I/O handling.

2.3.2 GAC (Global Assembly Cache) Folder

The following file is installed at the GAC folder where class library for .NET framework environment is installed. It is installed when installing the library for .NET during SDK installation.

2.3.3 (Installation Folder) \ Bin

Core files and sample execution files required in SDK execution are included.

UCSAPI40.dll / UCSAPICOM.dll / WSEngine.dll

Files identical to the ones installed at the Windows system32 folder.

Demo application

Several numbers of demo programs that can be used to test the functions of UCS SDK are included. All demo program sources are provided at the Samples folder.

2.3.4 (Installation Folder) \ dotNET

Class library files for .NET required in SDK execution are included.

UNIONCOMM.SDK.UCSAPI40.dll

The class library module for .NET. File identical to the one installed at GAC.

2.3.5 (Installation Folder) \ dotNET \ Setup

Files to install class library for .NET at GAC are included.

Setup.exe (UCSAPI40.NET_Setup.msi)

Class library installation file for .NET

2.3.6 (Installation Folder) \ Inc

UCSAPI.h

In case this file is included as the main header file of UCS SDK, UCSAPI_Basic.h, UCSAPI_Error.h and UCSAPI_Type.h files are included internally and automatically.

UCSAPI_Basic.h

The default data types used in UCS SDK are defined.

UCSAPI_Error.h

Error values used in UCSAPI40 module are defined.

UCSAPI_Type.h

Data type and structure information used in UCS SDK are defined.

2.3.7 (Installation Folder) \ Lib

The link library file for development at VC++ using SDK is included.

UCSAPI40.lib

The link library file created for VC++. It is used to link UCBioBSP.dll statically at VC++.

2.3.8 (Installation Folder) \ Samples

Sample source codes for each of the language are included in separate folders.

DLL

The sample code that can be developed using UCSAPI40.dll is included.

1) VC6: The sample created for Visual C++ 6.0 is included.

COM

The sample code that can be developed using UCSAPICOM.dll is included.

1) VB6: The sample created for Visual Basic 6.0 is included.

dotNET

The sample code that can be developed under Microsoft .NET environment using UNIONCOMM.SDK.UCSAPI40dll is included.

1) C#: The sample created for VisualStudio.NET 2005 and C# is included.

2.3.9 (Installation Folder) \ Skins

The skin resource file for each of the languages is included. Currently, only the English and Korean versions are included.

3. API Reference for DLL

Types and APIs to use a DLL module, UCSAPI40.dll, are described in the chapter.

3.1 Type definitions

3.1.1 Basic types

Basic types declared at UCSAPI_Basic are defined. Basic types are redefined for OS or CPU independent development. The following descriptions are based on the development with C++ under Windows.

UCSAPI_SINT8 / UCSAPI_SINT16 / UCBioAPI_SINT32

Signed 1byte / 2bytes / 4bytes value

UCSAPI_UINT8 / UCSAPI_UINT16 / UCBioAPI_UINT32

Unsigned 1byte / 2bytes / 4bytes value

UCSAPI_SINT / UCSAPI_UINT

Int/Unsigned int value that varies according to OS. It operates with 4bytes for 32bit OS and with 8bytes for 64bit OS.

UCSAPI_VOID_PTR

This type represents void*.

UCSAPI_BOOL

This type can have UCSAPI_FALSE(0) / UCBioAPI_TRUE(1) value. It is handled in the same way as int.

UCSAPI_CHAR / UCSAPI_CHAR_PTR

This type represents char and char*. 1byte character and character string value.

UCSAPI_NULL

This type represents NULL. It is defined with the value of ((void*)0).

UCSAPI_HWND

This type is the HWND value that represents Windows handle.

3.1.2 General types

Declaration is made at UCAPI_Type.h, and general types are defined.

UCSAPI_RETURN

Prototype:

typedef UCSAPI_UINT32 UCSAPI_RETURN;

Description:

Values returned by functions of UCSAPI SDK are defined. In general, error values of UCS SDK are included. Refer to the error definition for more information on error values.

UCSAPI_DATE_TIME_INFO

Prototype:

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

Description:

The structure that contains the date and time information

3.1.3 User information related types

Declaration is made at UCAPI_Type.h, and user information related types are defined.

UCSAPI_ACCESS_DATE_TYPE

Prototype:

```
typedef UCSAPI_UINT32          UCSAPI_ACCESS_DATE_TYPE;
```

Description:

Data types of the UCSAPI_ACCESS_DATE structure are defined. 3 data types can be defined for access period data; not used, allowed access period, and access restriction period. The following values are available.

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

UCSAPI_ACCESS_AUTHORITY

Prototype:

```
typedef struct ucsapi_access_authority
{
    UCSAPI_DATA_PTR             AccessGroup;
    UCSAPI_ACCESS_DATE_TYPE     AccessDateType;
    UCSAPI_ACCESS_DATE_PTR      AccessDate
} UCSAPI_ACCESS_AUTHORITY, UCSAPI_ACCESS_AUTHORITY_PTR;
```

Description:

The structure that contains the access rights information of the user. Each of values is described below.

AccessGroup:

The pointer on the structure that contains access rights group code information

AccessDateType:

The type of data that the UCSAPI_ACCESS_DATE structure has is designated.

AccessDate:

The pointer on the structure that contains access period information

UCSAPI_CARD_DATA

Prototype:

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

Description:

The structure that contains code information. Each of values is described below.

CardNum:

It designates the total number of RFID. RFID information corresponding to the number designated here are included as array.

RFID:

The pointer array of the structure that contains RFID information

UCSAPI_FINGER_DATA

Prototype:

```
typedef struct ucsapi_finger_data
{
    UCSAPI_UINT32          SecurityLevel;
    UCSAPI_BOOL            IsCheckSimilarFinger;
    UCSAPI_EXPORT_DATA_PTR ExportData;
} UCSAPI_FINGER_DATA, UCSAPI_FINGER_DATA_PTR;
```

Description:

The structure that contains fingerprint information. Each of values is described below.

SecurityLevel :

It designates the security level used during authentication.
Refer to the UCBioAPI_FIR_SECURITY_LEVEL definition of UCBioBSP SDK for available values.

IsCheckSimilarFinger :
When adding user fingerprint data to the terminal, It designates whether to check a similar fingerprint or not.
If this value is designated as true, the terminal checks if a similar fingerprint exists by comparing with the fingerprints of all registered users. If a similar fingerprint is detected, registration fails. As this flag can slow down user addition job by the terminal, performance may be degraded if there are a large number of registered users. It is used during UCSAPI_AddUserToTerminal call.

ExportData:
The pointer of the structure that contains converted template data
Refer to UCBioAPI_EXPORT_DATA structure of UCBioBSP SDK.

UCSAPI_AUTH_DATA

Prototype:

```
typedef struct ucsapi_auth_data
{
        UCSAPI_DATA_PTR         Password;
        UCSAPI_CARD_DATA_PTR    Card;
        UCSAPI_FINGER_DATA_PTR  Finger;
} UCSAPI_AUTH_DATA, UCSAPI_AUTH_DATA_PTR;
```

Description:
The structure that contains authentication information. Each of values is described below.

Password:
The pointer of the structure that contains password information

Card :
The pointer of the structure that contains card information

Finger:

The pointer of the structure that contains fingerprint information

UCSAPI_PICTURE_HEADER

Prototype:

```
typedef struct ucsapi_picture_header
{
        UCSAPI_UINT8        Format[4];      /* must be "jpg" */
        UCSAPI_UINT32       Length;         /* max length is 7 kbytes */
} UCSAPI_PICTURE_HEADER, UCSAPI_ PICTURE_HEADER_PTR;
```

Description:

The structure that contains the header information of picture data. Each of the values is described below.

Format :

It contains the format information of picture data. (Currently, only "JPG" format is supported.)

It designates the file extension value with the character string.

Length:

It has the size value of picture data. The maximum data that can be designated is 7KB.

UCSAPI_PICTURE_DATA

Prototype:

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

Description:

The structure that contains picture data information

Header :

It contains the header information of picture data.

Data :
The pointer of the buffer that contains image data in UCSAPI_PICTURE_HEADER format (Binary stream). The resolution of "JPG" data is 320*240.

UCSAPI_USER_COUNT

Prototype:

```
typedef struct ucsapi_user_count
{
        UCSAPI_UINT32        AdminNumber;
        UCSAPI_UINT32        UserNumber;
} UCSAPI_USER_COUNT, *UCSAPI_USER_COUNT_PTR;
```

Description:

The structure that contains the number of users registered in the terminal.
There are two types of users; administrator and general user.
After the UCSAPI_GetUserCountFromTerminal function called, it can be obtained from UCSAPI_CALLBACK_EVENT_GET_USER_COUNT event. Each of the values is described below.

AdminNumber:
It has the value of the number of registered administrators.

UserNumber :
It has the value of the number of registered general users.

UCSAPI_USER_INFO

Prototype:

```
typedef struct ucsapi_user_info
{
        UCSAPI_UINT32                UserID;
        UCSAPI_DATA_PTR              UserName;
        UCSAPI_DATA_PTR              UniqueID;
        UCSAPI_USER_PROPERTY         Property;
        UCSAPI_UINT8                 Reserved[3];
```

```
    UCSAPI_ACCESS_AUTHORITY_PTR   AccessAuthority;
} UCSAPI_USER_INFO, UCSAPI_ USER_INFO_PTR;
```

Description:

The structure that contains user information. Each of the values is described below.

UserID **:**

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

UserName **:**

The pointer of the structure that contains user name information. This value can be designated up to the size of UCSAPI_DATA_SIZE_USER_NAME.

UniqueID**:**

The pointer of the structure that contains unique ID (employee ID) information. This value can be used in place of UserID for user identification. It can be designated up to the size of UCSAPI_DATA_SIZE_UNIQUE_ID.

Property**:**

The structure that contains user property (authentication type and whether a user is an administrator or not) information

AccessAuthority:

The pointer of the structure that contains access rights information

UCSAPI_USER_DATA

Prototype:

```
typedef struct ucsapi_user_data
{
    UCSAPI_USER_INFO            UserInfo;
    UCSAPI_AUTH_DATA_PTR        AuthData;
    UCSAPI_PICTURE_DATA_PTR     PictureData;
} UCSAPI_USER_DATA, UCSAPI_ USER_DATA_PTR;
```

Description:

The structure that contains user data. It is used during UCSAPI_AddUserToTerminal call. Each of the values is described below.

UserInfo :
The structure that contains user information

AuthData :
The pointer of the structure that contains access rights data

PictureData :
The pointer of the structure that contains picture data

3.1.4 Log related types

Declaration is made at UCAPI_Type.h, and authentication log related types are defined.

UCSAPI_GET_LOG_TYPE

Prototype:

typedef UCSAPI_UINT32 UCSAPI_GET_LOG_TYPE;

Description:

To obtain log data from the terminal, three log types are to be defined.
The number of logs that can be stored internally varies according to terminal model. If the maximum storage capacity is exceeded, new logs are stored by deleting some of the existing records.
It is used during UCSAPI_GetAccessLogCountFromTerminal / UCSAPI_GetAccessLogFromTerminal call.

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

UCSAPI_ACCESS_LOG_DATA

Prototype:

```
typedef struct ucsapi_access_log_data
{
```

```
    UCSAPI_UINT32               UserID;
    UCSAPI_DATE_TIME_INFO       DataTime;
    UCSAPI_UINT8                AuthMode;
    UCSAPI_UINT8                AuthType;
    UCSAPI_UINT8                Reserved[2];
    UCSAPI_BOOL                 IsAuthorized;
    UCSAPI_DATA_PTR             RFID;
    UCSAPI_PICTURE_DATA_PTR     PictureData;
} UCSAPI_ACCESS_LOG_DATA, UCSAPI_ACCESS_LOG_DATA_PTR;
```

Description:

The structure that contains authentication log data

It can be obtained from the UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG event after UCSAPI_GetAccessLogFromTerminal is called. Each of the values is described below.

UserID **:**

It has the user ID value.

DateTime **:**

The structure that contains authentication time information

AuthMode**:**

It has the authentication mode value. Refer to UCSAPI_AUTH_MODE definition.

AuthType**:**

It has the authentication type value. Refer to UCSAPI_AUTH_TYPE definition.

IsAuthorized:

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

RFID:

The pointer of the structure that contains RFID data used during card authentication

PictureData:

The pointer of the structure that contains picture data taken during authentication. This value is available only for terminals that can take picture.

3.1.5 Callback related types

Declaration is made at UCAPI_Type.h, and callback event types are defined.

To notify data received from the terminal to the application program, SDK uses the callback function. The callback event consists of two parts; response to application program's request and request from the terminal.

UCSAPI_CALLBACK_EVENT_CONNECTED

Prototype:

#define UCSAPI_CALLBACK_EVENT_CONNECTED UCSAPI_CALLBACK_EVENT+1

Description:

When the terminal is connected to the server, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_DISCONNECTED

Prototype:

#define UCSAPI_CALLBACK_EVENT_DISCONNECTED UCSAPI_CALLBACK_EVENT+2

Description:

When the terminal is disconnected from the server, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_REALTIME_ACCESS_LOG

Prototype:

#define UCSAPI_CALLBACK_EVENT_REALTIME_ACCESS_LOG UCSAPI_CALLBACK_EVENT+3

Description:

When the terminal operates in S/N mode and the terminal implemented user authentication, SDK notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG

Prototype:

#define UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG UCSAPI_CALLBACK_EVENT+4

Description:

In response to the application program's request for authentication logs stored at the terminal, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG_COUNT

Prototype:

#define UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG_COUNT UCSAPI_CALLBACK_EVENT+5

Description:

In response to the application program's request for the number of authentication logs stored at the terminal, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_ADD_USER

Prototype:

#define UCSAPI_CALLBACK_EVENT_ADD_USER UCSAPI_CALLBACK_EVENT+10

Description:

In response to the user addition request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_DELETE_USER

Prototype:

#define UCSAPI_CALLBACK_EVENT_DELETE_USER UCSAPI_CALLBACK_EVENT+11

Description:

In response to the user deletion request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_DELETE_ALL_USER

Prototype:

#define UCSAPI_CALLBACK_EVENT_DELETE_ALL_USER UCSAPI_CALLBACK_EVENT+12

Description:

In response to the all user deletion request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_GET_USER_COUNT

Prototype:

#define UCSAPI_CALLBACK_EVENT_GET_USER_COUNT UCSAPI_CALLBACK_EVENT+13

Description:

In response to the application program's request for the number of users stored at the terminal, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_GET_USER_INFO_LIST

Prototype:

#define UCSAPI_CALLBACK_EVENT_GET_USER_INFO_LIST UCSAPI_CALLBACK_EVENT+14

Description:

In response to the application program's request for user lists stored at the terminal, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_GET_USER_DATA

Prototype:

#define UCSAPI_CALLBACK_EVENT_GET_USER_DATA UCSAPI_CALLBACK_EVENT+15

Description:

In response to the application program's request for user data stored at the terminal, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO

Prototype:

#define UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO UCSAPI_CALLBACK_EVENT+20

Description:

To obtain information (authentication type) required in 1:1 server authentication, the terminal notifies this event to the application program. Then, the application program needs to send the authentication type information to the terminal. This event is generated only when authentication is implemented at the server, and it is always issued before the next event.

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1,

UCSAPI_CALLBACK_EVENT_VERIFY_CARD,

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1

Prototype:

#define UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1 UCSAPI_CALLBACK_EVENT+21

Description:

During 1:1 fingerprint authentication, the terminal notifies this event to the application program. Then, the application program needs to send the authentication results to the terminal. This event is generated only when authentication is implemented at the server.

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N

Prototype:

#define UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N UCSAPI_CALLBACK_EVENT+22

Description:

During 1:N fingerprint authentication, the terminal notifies this event to the application program. Then, the application program needs to send the authentication results to the terminal. This event is generated only when authentication is implemented at the server.

UCSAPI_CALLBACK_EVENT_VERIFY_CARD

Prototype:

#define UCSAPI_CALLBACK_EVENT_VERIFY_CARD UCSAPI_CALLBACK_EVENT+23

Description:

During card authentication, the terminal notifies this event to the application program. Then, the application program needs to send the authentication results to the terminal. This event is generated only when authentication is implemented at the server.

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD

Prototype:

#define UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD UCSAPI_CALLBACK_EVENT+24

Description:

During password authentication, the terminal notifies this event to the application program. Then, the application program needs to send the authentication results to the terminal. This event is generated only when authentication is implemented at the server.

UCSAPI_CALLBACK_EVENT_GET_TERMINAL_OPTION

Prototype:

#define UCSAPI_CALLBACK_EVENT_GET_TERMINAL_OPTON UCSAPI_CALLBACK_EVENT+30

Description:

In response to the terminal option setting information request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_SET_TERMINAL_OPTION

Prototype:

#define UCSAPI_CALLBACK_EVENT_SET_TERMINAL_OPTION UCSAPI_CALLBACK_EVENT+31

Description:

In response to the terminal option setting request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_FW_UPGRADING

Prototype:

#define UCSAPI_CALLBACK_EVENT_FW_UPGRADING UCSAPI_CALLBACK_EVENT+80

Description:

In response to the firmware upgrade request by the application program, the SDK module notifies this event to the application program. It is the event to notify the upgrade progress information to the application program.

UCSAPI_CALLBACK_EVENT_FW_UPGRADE

Prototype:

#define UCSAPI_CALLBACK_EVENT_FW_UPGRADED UCSAPI_CALLBACK_EVENT+81

Description:

In response to the firmware upgrade request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_FW_VERSION

Prototype:

#define UCSAPI_CALLBACK_EVENT_FW_VERSION UCSAPI_CALLBACK_EVENT+82

Description:

In response to the firmware version request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_OPEN_DOOR

Prototype:

#define UCSAPI_CALLBACK_EVENT_OPEN_DOOR UCSAPI_CALLBACK_EVENT+90

Description:

In response to the temporary door open request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_TERMINAL_STATUS

Prototype:

#define UCSAPI_CALLBACK_EVENT_TERMINAL_STATUS UCSAPI_CALLBACK_EVENT+91

Description:

The SDK module periodically or immediately notifies the server about the status information of the terminal and devices connected to the terminal when the status is changed.

UCSAPI_CALLBACK_EVENT_PICTURE_LOG

Prototype:

#define UCSAPI_CALLBACK_EVENT_PICTURE_LOG UCSAPI_CALLBACK_EVENT+100

Description:

When the picture log is received from the terminal, the SDK module notifies the picture log data to the application program. If authentication is implemented with the terminal authentication type, the picture log is sent to the application program along with the UCSAPI_CALLBACK_EVENT_REALTIME_ACCESS_LOG event. But, if authentication is implemented with the server authentication type, the terminal notifies the picture log to the application program separately after receiving the authentication results from the application program.

UCSAPI_CALLBACK_EVENT_ANTIPASSBACK

Prototype:

#define UCSAPI_CALLBACK_EVENT_ANTIPASSBACK UCSAPI_CALLBACK_EVENT+101

Description:

In case that the anti-passback option is not set, the terminal notifies this event to the application program to obtain the user's current anti-passback status during user authentication. Then, the application program needs to immediately send user's anti-passback status to the terminal. This event is generated only when authentication is implemented at the terminal.

UCSAPI_CALLBACK_EVENT_SET_ACCESS_CONTROL_DATA

Prototype:

#define UCSAPI_CALLBACK_EVENT_SET_ACCESS_CONTROL_DATA
UCSAPI_CALLBACK_EVENT+102

Description:

In response to the access rights setting request by the application program, the SDK module notifies this event to the application program.

UCSAPI_CALLBACK_EVENT_TYPE

Prototype:

typedef UCSAPI_UINT32 UCSAPI_CALLBACK_EVENT_TYPE

Description:

This type defines the second element value of UCSAPI_CALLBACK_EVENT_HANDLER.

UCSAPI_CALLBACK_EVENT_HANDLER

Prototype:

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

Description:

This type defines the callback function to receive the event generated from the terminal.

UCSAPI_CALLBACK_PARAM_0

Prototype:

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

Description:

The structure passed as the third element of UCSAPI_CALLBACK_EVENT_HANDLER. Each of the values is described below.

ClientID :

ID of the client that requested the job. (It is used in client/server model development.)

ErrorCode :

It contains the value on errors generated from the executed job.

The value of 0 represents success, while all other values represent failure.

Progress :

The structure that contains the progress information of the executed job.

It can be obtained after UCSAPI_GetUserInfoListFromTerminal / UCSAPI_GetAccessLogFromTerminal / USCAPI_UpgradeFirmwareToTerminal functions are called.

UCSAPI_CALLBACK_DATA_TYPE

Prototype:

```
typedef UCSAPI_UINT32 UCSAPI_CALLBACK_DATA_TYPE;
```

Description:

Data types of the UCSAPI_CALLBACK_PARAM_1 structure are defined.

```
#define UCSAPI_CALLBACK_DATA_TYPE_USER_INFO          0
#define UCSAPI_CALLBACK_DATA_TYPE_USER_DATA          1
#define UCSAPI_CALLBACK_DATA_TYPE_ACCESS_LOG         2
```

UCSAPI_CALLBACK_PARAM_1

Prototype:

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

Description:

The structure passed as the fourth element of UCSAPI_CALLBACK_EVENT_HANDLER. Each of the values is described below.

DataType:

The type of data that this structure has is designated. Refer to UCSAPI_CALLBACK_DATA_TYPE.

Data:

The union structure that designates real data. Values of UserInfo, UserData and AccessLog can be used by storing them with a single identical address pointer.

UCSAPI_PROGRESS_INFO

Prototype:

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

Description:

The structure that contains the progress information of the executed job. When notifying several records to the application program, UCS SDK includes progress information in this structure and notifies it to the application program along with the UCSAPI_CALLBACK_PARAM_0 structure.

It can be obtained after UCSAPI_GetUserInfoListFromTerminal / UCSAPI_GetAccessLogFromTerminal / USCAPI_UpgradeFirmwareToTerminal functions are called. Each of the values is described below.

CurrentIndex:

The index of the record currently in transmission

TotalNumber :

The total number of records to be sent

3.1.6 Access control setting related types

Declaration is made at UCAPI_Type.h, and terminal access control related types are defined.

UCSAPI_TIMEZONE

Prototype:

```
typedef struct ucsapi_timezone
{
```

```
    UCSAPI_TIME_HH_MM       StartTime;
    UCSAPI_TIME_HH_MM       EndTime;
} UCSAPI_TIMEZONE, * UCSAPI_TIMEZONE_PTR;
```

Description:

The structure that contains time zone information

StartTime/ EndTime:

The structure that contains time information from start to end

UCSAPI_ACCESS_TIMEZONE

Prototype:

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

Description:

The structure that contains information on the time zone allowed for access during a day Up to 12 time zones can be designated into a single time code. Each of the values is described below.

Code:

As the identifier code value of the time zone, it is a character string of fixed UCSAPI_DATA_SIZE_CODE size.

Zone:

The structure array that contains time zone information

UCSAPI_ACCESS_TIMEZONE_DATA

Prototype:

```
typedef struct ucsapi_access_timezone_data
{
```

```
    UCSAPI_UINT32                   TimezoneNum;
    UCSAPI_ACCESS_TIMEZONE          Timezone[UCSAPI_ACCESS_TIMEZONE_MAX];
} UCSAPI_ACCESS_TIMEZONE_DATA, * UCSAPI_ACCESS_TIMEZONE_DATA_PTR;
```

Description:

The structure that contains the allowed access time zone data. Up to 128 time zone code information can be designated.

TimezoneNum:

It designates the total number of allowed access time zone codes. Time zone information corresponding to the number designated here are included in the form of an array.

Timezone:

The structure array that contains the allowed access time zone code information

UCSAPI_ACCESS_HOLIDAY

Prototype:

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

Description:

The structure that contains holiday information

Up to 32 holidays can be designated into the holiday code. Each of the values is described below.

Code:

As the identifier code value of a holiday, it is a character string of fixed UCSAPI_DATA_SIZE_CODE size.

Date:

The structure array that contains holiday information

UCSAPI_ACCESS_HOLIDAY_DATA

Prototype:

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

Description:

The structure that contains holiday data.

Up to 64 holiday code data can be designated. Each of the values is described below.

HolidayNum:

It designates the total number of holiday codes. Holiday information corresponding to the number designated here are included in the form of an array.

Holiday:

The structure array that contains holiday code information

UCSAPI_ACCESS_TIMEZONE_CODE

Prototype:

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

Description:

The structure that contains the allowed the access time zone code for each day of the

week. Each of the values is described below.

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

They have the allowed access time zone code value for each day of the week to be used during authentication.

Hol:

It has the allowed access time zone code value to be applied to the holiday during authentication.

UCSAPI_ACCESS_TIME

Prototype:

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

Description:

The structure that contains the allowed access time information. Each of the values is described below.

Code:

As the identifier code value of allowed access time, it is a character string of fixed UCSAPI_DATA_SIZE_CODE size.

Timezone:

It has the allowed access time zone code for each day of the week.

Holiday:

It has the holiday code value to be used in the allowed access time code.

The time zone designated at Hol of the UCSAPI_ACCESS_TIMEZONE_CODE structure is applied to the holiday code designated here.

UCSAPI_ACCESS_TIME_DATA

Prototype:

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

Description:

The structure that contains the allowed access time data.

Up to 128 allowed access time code information can be designated. Each of the values is described below.

AccessTimeNum:

It designates the total number of allowed access time codes. AccessTime information corresponding to the number designated here are included in the form of an array.

AccessTime:

The structure array that contains the allowed authentication time code information

UCSAPI_ACCESS_GROUP

Prototype:

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

Description:

The structure that contains the access group code information. Each of the values is described below.

Up to 4 allowed access time codes can be designated to the access group code.

Code:

As the identifier code value of the access group, it is a character string of fixed UCSAPI_DATA_SIZE_CODE size.

AccessTime1 / AccessTime2 / AccessTime3 / AccessTime4:

They contain the allowed access time code information to be used in the access group.

UCSAPI_ACCESS_GROUP_DATA

Prototype:

```
typedef struct ucsapi_access_group_data
{
        UCSAPI_UINT32           AccessGroupNum;
        UCSAPI_ACCESS_GROUP     AccessGroup[UCSAPI_ACCESS_GROUP_MAX];
} UCSAPI_ACCESS_GROUP_DATA, * UCSAPI_ACCESS_GROUP_DATA_PTR;
```

Description:

The structure that contains access group data. Up to 128 access group code information can be designated.

Each of the values is described below.

AccessGroupNum:

It designates the total number of access group codes. Access group information corresponding to the number designated here are included in the form of an array.

AccessGroup:

The structure array that contains the access group code information

UCSAPI_ACCESS_CONTROL_DATA_TYPE

Prototype:

```
typedef UCSAPI_UINT32           UCSAPI_ACCESS_CONTROL_DATA_TYPE

#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_TIMEZONE        0
#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_HOLIDAY         1
#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_TIME            2
```

```
#define UCSAPI_ACCESS_CONTROL_DATA_TYPE_GROUP                3
```

Description:

It designates the data type that the UCSAPI_ACCESS_CONTROL_DATA structure has.

UCSAPI_ACCESS_CONTROL_DATA

Prototype:

```
typedef struct ucsapi_access_control_data
{
        UCSAPI_ACCESS_CONTROL_DATA_TYPE         DataType;
        union {
                UCSAPI_ACCESS_TIMEZONE_DATA_PTR         Timezone;
                UCSAPI_ACCESS_HOLIDAY_DATA_PTR          Holiday;
                UCSAPI_ACCESS_TIME_DATA_PTR             AccessTime;
                UCSAPI_ACCESS_GROUP_DATA_PTR            AccessGroup;
        } Data;
} UCSAPI_ACCESS_CONTROL_DATA, * UCSAPI_ACCESS_CONTROL_DATA_PTR;
```

Description:

The structure that contains the access control information. Values of Timezone, Holiday, AccessTime and AccesGroup can be used by storing them with a single identical address pointer.

It is used in the UCSAPI_SetAccessControlDataToTerminal function.

Each of the values is described below.

DataType:

The type of data that this structure has is designated. Refer to UCSAPI_ACCESS_CONTROL_DATA_TYPE.

3.1.7 Authentication related types

Declaration is made at UCAPI_Type.h, and user authentication related types are defined.

UCSAPI_AUTH_TYPE

Prototype:

typedef UCSAPI_UINT32 UCSAPI_AUTH_TYPE;

Description:

The authentication type is defined during authentication.

```
#define UCSAPI_AUTH_TYPE_FINGER_1_TO_N          0
#define UCSAPI_AUTH_TYPE_FINGER_1_TO_1          1
#define UCSAPI_AUTH_TYPE_FINGER_CARD            2
#define UCSAPI_AUTH_TYPE_CARD                   3
#define UCSAPI_AUTH_TYPE_PASSWORD               4
```

UCSAPI_AUTH_MODE

Prototype:

typedef UCSAPI_UINT32 UCSAPI_AUTH_MODE;

Description:

The authentication mode is defined during user authentication. The authentication mode defines its purpose during authentication. In general, it can be applied when the terminal is used for time/attendance management.

```
#define UCSAPI_AUTH_MODE_ATTENDANCE     0
#define UCSAPI_AUTH_MODE_LEAVE          1
#define UCSAPI_AUTH_MODE_NORMAL         2
#define UCSAPI_AUTH_MODE_OUT            3
#define UCSAPI_AUTH_MODE_RETURN         4
```

UCSAPI_INPUT_DATA_CARD

Prototype:

```
typedef struct ucsapi_input_data_card
{
        UCSAPI_UINT32                   AuthMode;
```

```
    UCSAPI_DATA                 RFID;
} UCSAPI_INPUT_DATA_CARD, *UCSAPI_INPUT_DATA_CARD_PTR;
```

Description:

The structure that contains the information entered during card authentication at the terminal. Each of the values is described below.

AuthMode:

It has the authentication mode value entered at the terminal. Refer to UCSAPI_AUTH_MODE.

RFID:

The structure that contains the RFID information entered at the terminal.

UCSAPI_INPUT_DATA_PASSWORD

Prototype:

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

Description:

The structure that contains the information entered during password authentication at the terminal. Each of the values is described below.

UserID:

It has user ID information.

AuthMode:

It has the authentication mode value. Refer to UCSAPI_AUTH_MODE.

Password:

The structure that contains password information

UCSAPI_INPUT_DATA_FINGER_1_TO_1

Prototype:

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

Description:

The structure that contains the information entered during 1:1 fingerprint authentication at the terminal. Each of the values is described below.

UserID:

It has user ID information.

AuthMode:

It has the authentication mode value. Refer to UCSAPI_AUTH_MODE.

SecurityLevel:

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

Refer to the UCBioAPI_FIR_SECURITY_LEVEL definition of UCBioBSP SDK for available values.

Finger:

The structure that contains fingerprint information

UCSAPI_INPUT_DATA_FINGER_1_TO_N

Prototype:

```
typedef struct ucsapi_input_data_finger_1_to_n
{
        UCSAPI_UINT32           AuthMode;
        UCSAPI_UINT32           SecurityLevel;
        UCSAPI_UINT32           InputIDLength;
```

```
    UCSAPI_DATA                 Finger;
} UCSAPI_INPUT_DATA_FINGER_1_TO_N, *UCSAPI_INPUT_DATA_FINGER_1_TO_N_PTR;
```

Description:

The structure that contains the information entered during 1:N fingerprint authentication at the terminal. Each of the values is described below.

UserID:

It has user ID information.

AuthMode:

It has the authentication mode value. Refer to UCSAPI_AUTH_MODE.

SecurityLevel:

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

InputIDLength:

It has the length value of ID entered at the terminal. This value can be used to improve the speed of 1:N fingerprint authentication by reducing the authentication range. If the UserID value is 5 and the InputIDLength value is 3 in case the ID range of the registered user is 0001~1000, then the authentication ID range becomes 0500~1000.

Finger:

The structure that contains the fingerprint information entered at the terminal.

UCSAPI_INPUT_DATA_TYPE

Prototype:

```
typedef UCSAPI_UINT32       UCSAPI_INPUT_DATA_TYPE;

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

Description:

The type of data that the UCSAPI_INPUT_DATA_TYPE structure has is designated.

UCSAPI_INPUT_DATA

Prototype:

```
typedef struct ucsapi_input_data
{
        UCSAPI_ANTIPASSBACK_LEVEL       AntipassbackLevel;
        UCSAPI_INPUT_DATA_TYPE          DataType;
        Union {
                UCSAPI_INPUT_DATA_FINGER_1_TO_1_PTR         Finger1To1;
                UCSAPI_INPUT_DATA_FINGER_1_TO_N_PTR         Finger1ToN;
                UCSAPI_INPUT_DATA_CARD_PTR                  Card;
                UCSAPI_INPUT_DATA_PASSWORD_PTR              Password;
        } Data;
} UCSAPI_INPUT_DATA, *UCSAPI_INPUT_DATA_PTR;
```

Description:

The structure that contains the information entered at the terminal during user authentication. Authentication request can be made to the application program by storing the input information at this structure. Each of the values is described below.

AntipassbackLevel:

It has the anti-passback level set up at the terminal. The application program can refer to this value when implementing the anti-passback function.

DataType:

The type of data that this structure has is designated. Refer to UCSAPI_INPUT_DATA_TYPE.

Data:

The union structure that designates real data. Values of Finger1To1, Finger1ToN, Card and Password are used by storing them with a single identical address pointer.

UCSAPI_INPUT_ID_TYPE

Prototype:

```
typedef UCSAPI_UINT32          UCSAPI_INPUT_ID_TYPE;

#define UCSAPI_INPUT_ID_TYPE_USER_ID          0
#define UCSAPI_INPUT_ID_TYPE_UNIQUE_ID        1
#define UCSAPI_INPUT_ID_TYPE_RFID             2
```

Description:

The type of ID entered at the terminal is designated. The ID type entered during 1:1 authentication can be changed at the terminal option settings. The default value is the UCSAPI_INPUT_ID_TYPE_USER_ID type. The maximum size of the user ID value is 8 digits. If the maximum number of digits is exceeded, the use of the UCSAPI_INPUT_ID_TYPE_UNIQUE_ID type increases the maximum size to 20 digits.

UCSAPI_INPUT_ID_DATA

Prototype:

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

Description:

The structure that contains the ID information entered at the terminal during user authentication at the server. Each of the values is described below.

DataType:

The type of data that this structure has is designated. Refer to UCSAPI_INPUT_ID_TYPE.

Data:

The union structure that designates real data. Values of UserID, UniqueID and RFID can be used by storing them with a single identical address pointer.

UCSAPI_AUTH_INFO

Prototype:

```
typedef struct ucsapi_auth_info
{
    UCSAPI_UINT32           UserID;
    UCSAPI_BOOL             IsAccessibility;
    UCSAPI_USER_PROPERTY    Property;
    UCSAPI_UINT32           ErrorCode;
} UCSAPI_AUTH_INFO, *UCSAPI_AUTH_INFO_PTR;
```

Description:

The structure that contains the user's authentication information. It is used in the UCSAPI_SendAuthInfoToTerminal function.

Each of the values is described below.

UserID:

It has the user's ID value.

IsAccessibility:

It has the value on whether the user has authentication rights or not.

Property:

The structure that contains user property (authentication type and administrator) information

ErrorCode:

In case the user does not have authentication rights, it has the corresponding error code value. Refer to error code table.

UCSAPI_AUTH_RESULT

Prototype:

```
typedef struct ucsapi_auth_result
{
```

```
    UCSAPI_UINT32               UserID;
    UCSAPI_BOOL                 IsAuthorized;
    UCSAPI_BOOL                 IsVistor;
    UCSAPI_DATE_TIME_INFO       AuthorizedTime;
    UCSAPI_UINT32               ErrorCode;
} UCSAPI_INPUT_DATA, *UCSAPI_INPUT_DATA_PTR;
```

Description:

The structure that contains the user's authentication result information.

It is used in the UCSAPI_SendAuthResultToTerminal function.

UserID:

It has the ID value of the authenticated user or the user who attempted authentication.

IsAuthorized:

It has the authentication result value.

IsVisitor:

It has the value on whether the authenticated user is a visitor or not.

AuthorizedTime:

The structure that contains authentication time information

ErrorCode:

It has the error code value in case of authentication failure. Refer to the error code table.

3.1.8 Terminal option setting related types

Declaration is made at UCAPI_Type.h, and terminal option setting related types are defined.

UCSAPI_TERMINAL_TIMEZONE

Prototype:

```
typedef struct ucsapi_terminal_timezone
{
```

```
    UCSAPI_UINT8        IsUsed;
    UCSAPI_UINT8        StartHour;
    UCSAPI_UINT8        StartMin;
    UCSAPI_UINT8        EndHour;
    UCSAPI_UINT8        EndMin;
} UCSAPI_TERMINAL_TIMEZONE, * UCSAPI_TERMINAL_TIMEZONE_PTR;
```

Description:

The structure that contains the time information used at the lock/open schedule of the terminal. Each of the values is described below.

IsUsed :

It has information on whether the value that the UCSAPI_TERMINAL_TIMEZONE structure has is valid or not.

StartHour / StartMin:

It contains the start time information.

EndHour / EndMin :

It contains the end time information.

UCSAPI_TERMINAL_DAY_SCHEDULE

Prototype:

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

Description:

The structure that contains terminal's lock/open schedule information for each day of the

week.

Regarding the schedule for each day of the week, up to three lock/open time zones per day can be designated. Each of the values is described below.

Lock1 / Lock2 / Lock3:

They have the time zone value to lock the terminal during a day.

Open1 / Open2 / Open3:

They have the time zone value to open the terminal during a day.

UCSAPI_HOLIDAY_TYPE

Prototype:

```
typedef UCSAPI_UINT8            UCSAPI_HOLIDAY_TYPE;

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

Description:

The holiday type to be used in the lock/open schedule of the terminal is designated. Up to 3 holiday types can be designated. Each of the values is described below.

UCSAPI_TERMINAL_HOLIDAY_INFO

Prototype:

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

Description:

The structure that contains holiday information

Month/Day:

These structures contain the holiday's date information.

Holidaytype:

It has the holiday type value. Refer to UCSAPI_HOLIDAY_TYPE.

UCSAPI_TERMINAL_SCHEDULE

Prototype:

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

Description:

The structure that contains the terminal's lock/open schedule information for each day of the week.

Up to 100 holidays can be set up. The holiday type can be designated at Holiday1, Holiday2 and Holiday3. Each of the values is described below.

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

The structure that contains the lock/open schedule information for each day of the week

Holiday1 / Holiday2 / Holiday3:

The structure that contains the lock/open schedule information for each holiday type

Holidays:

The structure that contains holiday information. One schedule from Holiday1, Holiday2 and Holiday3 is applied to each holiday.

UCSAPI_SECURITY_LEVEL

Prototype:

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

Description:

The structure that contains the security level information to be used during fingerprint authentication. It can have any of the following values.

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

Verify:

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

Identify

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

UCSAPI_ANTIPASSBACK_LEVEL

Prototype:

```
typedef UCSAPI_UINT32           UCSAPI_ANTIPASSBACK_LEVEL;

#define UCSAPI_ANTIPASSBACK_LEVEL_NOT_USE                   0
#define UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTION_ALLOW  1
#define UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTION_PROHIBIT 2
```

Description:

It has the anti-passback level value set up at the terminal. To use the anti-passback function, the terminal needs to be set as UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTION_ALLOW or UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTION_PROHIBIT.

UCSAPI_ANTIPASSBACK_LEVEL_NOT_USE:

Anti-passback not used

UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTION_ALLOW:

Access allowed when connection to the server is disabled

UCSAPI_ANTIPASSBACK_LEVEL_WHEN_DISCONNECTION_PROHIBIT:

Access not allowed when connection to the server is disabled

UCSAPI_NETWORK_INFO

Prototype:

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

Description:

The structure that contains the terminal's network setting information

NetworkType :

It has the type value of IP address. The static IP is supported for the value of 0, while the dynamic IP is supported for the value of 1.

IP :

The array of the buffer that contains the IP address value of the terminal

Subnet :

The array of the buffer that contains the subnet mask value

Gateway :

The array of the buffer that contains the gateway address value

UCSAPI_SERVER_INFO

Prototype:

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

Description:

The structure that contains the network information for the terminal to connect to the server. Each of the values is described below.

IP :

The buffer array that contains the address value of server IP

Port :

It has the socket port value for connection to the server.

UCSAPI_TERMINAL_OPTION_FLAG

Prototype:

```
typedef struct ucsapi_terminal_option_flag
{
        UCSAPI_UINT32       SecurityLevel       :1;
        UCSAPI_UINT32       InputIDLength       :1;
        UCSAPI_UINT32       AutoEnterKey        :1;
        UCSAPI_UINT32       Sound               :1;
```

```
    UCSAPI_UINT32       Authentication      :1;
    UCSAPI_UINT32       Application         :1;
    UCSAPI_UINT32       Antipassback        :1;
    UCSAPI_UINT32       Network             :1;
    UCSAPI_UINT32       Server              :1;
    UCSAPI_UINT32       InputIDType         :1;
    UCSAPI_UINT32       AccessLevel         :1;
    UCSAPI_UINT32       PrintText           :1;
    UCSAPI_UINT32       Schedule            :1;
} UCSAPI_TERMINAL_OPTION_FLAG, *UCSAPI_TERMINAL_OPTION_FLAG_PTR;
```

Description:

It has the reference flag on each item of the UCSAPI_TERMINAL_OPTION structure. Only when the flag value of an item is true, the value of that item can be referenced.

Refer to the UCSAPI_TERMINAL_OPTION structure for description on each item.

UCSAPI_TERMINAL_OPTION

Prototype:

```
typedef struct ucsapi_terminal_option
{
    UCSAPI_TERMINAL_OPTION_FLAG  Flags;
    UCSAPI_SECURITY_LEVEL        SecurityLevel;
    UCSAPI_UINT8                 InputIDLength;
    UCSAPI_UINT8                 AutoEnterKey;
    UCSAPI_UINT8                 Sound;
    UCSAPI_UINT8                 Authentication;
    UCSAPI_UINT8                 Application;
    UCSAPI_UINT8                 Antipassback;
    UCSAPI_NETWOARK_INFO         Network;
    UCSAPI_SERVER_INFO           Server;
    UCSAPI_UINT8                 InputIDType;
    UCSAPI_UINT8                 AccessLevel;
    UCSAPI_UINT8                 PrintText[32];
    UCSAPI_TERMINAL_SCHEDULE     Schedule;
} UCSAPI_TERMINAL_OPTION, *UCSAPI_TERMINAL_OPTION_PTR;
```

Description:

The structure that contains the option setting value of the terminal.

It is used in the UCSAPI_SetOptionToTerminal/UCSAPI_GetOptionFromTerminal function. Each of the values is described below.

Flags :

It has the reference flag value on each item of the structure.

To set up the terminal's option items using the UCSAPI_SetOptionToTerminal function, designate the flag value of an item as true and designate the value of that item.

SecurityLevel :

It designates the security level to be used during authentication.

Refer to the UCBioAPI_FIR_SECURITY_LEVEL definition of UCBioBSP SDK for available values.

InputIDLength :

It has the length value of ID entered at the terminal. The maximum length of 8 digits can be designated in case of using UserID while the maximum of length of 20 digits in case of using UniqueID.

AutoEnterKey :

It has the value on whether the terminal can use the automatic Enter key function or not. When key input corresponds to InputIDLength, this function enters the "Enter" key automatically.

Sound :

It has the sound volume value of the terminal. The range of the volume value that can be designated is 0~20. To set the terminal's sound to mute, 0 is designated.

Authentication :

It has the authentication type value of the terminal.

Refer to "Terminal Authentication Type" in Section 1.6 Terminology Description for available values.

Application.

It has the mode value of the terminal program. The terminal can be used as access control, time/attendance and drinking water management function. Refer to "Terminal Program Mode" in Section 1.6 Terminology Description for available values.

Antipassback:
It has the anti-passback level value of the terminal.
Refer to the UCSAPI_ANTIPASSBACK_LEVEL definition for available values.

Network:
The structure that contains terminal's network information

Server:
The structure that contains network information for the terminal to connect to the server

InputIDType:
It has the type value of ID entered at the terminal during authentication. The following values are available.

0 – UserID
1 – UniqueID

AccessLevel:
It has the access level value. This is the function that restricts the authentication type, allowing only the designated types for authentication. The following values are available. The default value is 0.

0 – No restriction
1 – Only fingerprint and password authentication allowed

PrintText:
It is the buffer array that contains character strings to be printed at the drinking water printer connected to the terminal.
This value can be used when the drinking water printer is connected to the terminal.

Schedule : The structure that contains lock/open schedule information

3.1.9 Monitoring related types

Declaration is made at UCAPI_Type.h, and monitoring related types are defined.

UCSAPI_TERMINAL_STATUS

Prototype:

```
typedef struct ucsapi_terminal_status
{
        UCSAPI_UINT32       Terminal;
        UCSAPI_UINT32       Door;
        UCSAPI_UINT32       Cover;
} UCSAPI_TERMINAL_STATUS, *UCSAPI_TERMINAL_STATUS_PTR;
```

Description:

The structure that contains terminal's status value. Each of the values is described below.

Terminal:

It has the lock status value of the terminal. The following values are available.

0 – UnLock

1 – Lock

Door:

It has the locking device status value of the terminal. This value is supported only for locking devices with the monitoring function. The following values are available.

0 – Close

1 – Open

2 – Not used

Cover:

It has the cover status value of the terminal. The following values are available.

0 – Close

1 – Open

3.3 API References

Definitions on various APIs used in UCS SDK, instructions on function use and elements are described.

3.3.1 Initialize API

APIs to start/stop UCS SDK are described.

UCSAPI_ServerStart

Prototype:

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

Description:

This API initializes UCSAPI modules and implements server functions.

Parameters:

MaxTerminal::

This parameter defines the maximum number of terminals that can be connected. SDK can improve speed by assigning internal memory capacity in advance according to the maximum number of terminals. If the number of connected terminals exceeds the maximum number, memory is increased automatically to increase the number of connections.

Port:

The communication port for terminal connection. The server is in standby mode for terminal's connection to the designated port. The default value is 9870. If this value is changed, the terminal's port value also needs to be changed.

CallBackEventFunction:

The pointer on the callback function to notify an event to an application program

Returns:

UCSAPIERR_NONE

UCSAPIERR_FUNCTION_FAILED

Callback:

UCSAPI_CALLBACK_EVENT_CONNECTED

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0 CallBack0

lParam:

UCSAPI_UINT8 TerminalIP[4]:

The array of the buffer that has the terminal's IP address

UCSAPI_ServerStop

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_ServerStop();

Description:

This API disconnects all connected terminals and stops server functions.

Parameters:

N/A

Returns:

UCSAPIERR_NONE

Callback:

N/A

Callback Parameters:

N/A

3.3.2 Terminal User Management API

APIs that can manage terminal users are described.

UCSAPI_AddUserToTerminal

Prototype:

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

Description:

This API sends the user information to a designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

IsOverwrite:

This parameter designates whether to overwrite an already registered user or not. The default value is 1.

pUserData:

The pointer of the structure that contains user data

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL
UCSAPIERR_USER_NAME_SIZE
UCSAPIERR_UNIQUE_ID_SIZE
UCSAPIERR_INVALID_SECURITY_LEVEL
UCSAPIERR_INVALID_PARAMETER
UCSAPIERR_CODE_SIZE
UCSAPIERR_PASSWORD_SIZE
UCSAPIERR_MAX_CARD_NUMBER
UCSAPIERR_MAX_FINGER_NUMBER
UCSAPIERR_PICTURE_SIZE

Callback:

UCSAPI_CALLBACK_EVENT_ADD_USER

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

UCSAPI_UINT32 UserID;

※ Note

In case of sending a multiple numbers of user data to the terminal using UCSAPI_AddUserToTerminal function, the UCSAPI_CALLBACK_EVENT_ADD_USER event must be checked after UCSAPI_AddUserToTerminal is called. The next user data is sent only after transmission is processed normally.

UCSAPI_DeleteUserFromTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_DeleteUserFromTerminal(

IN UCSAPI_INT32 ClientID,

IN UCSAPI_INT32 TerminalID,

IN UCSAPI_INT32 UserID);

Description:

This API deletes the user data from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

UserID:

ID of a user to delete

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_DELETE_USER

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam :

UCSAPI_UINT32 UserID;

※ Note

In case of deleting several users of the terminal using UCSAPI_DeleteUserFromTerminal function, the UCSAPI_CALLBACK_EVENT_DELETE_USER event must be checked after UCSAPI_DeleteUserFromTerminal is called. The next user is deleted only after deletion is processed normally.

UCSAPI_DeleteAllUserFromTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_DeleteAllUserFromTerminal(

IN UCSAPI_INT32 ClientID,

IN UCSAPI_INT32 TerminalID);

Description:

This API deletes all user data from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_DELETE_ALL_USER

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

N/A

UCSAPI_GetUserCountFromTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_GetUserCountFromTerminal(

IN UCSAPI_INT32 ClientID,

IN UCSAPI_INT32 TerminalID);

Description:

This API obtains the number of registered users from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_USER_COUNT

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

UCSAPI_UINT32 nUserCount

This parameter contains the number of users.

UCSAPI_GetUserInfoListFromTerminal

Prototype:

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

Description:

This API obtains the list of all registered user information from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_USER_INFO_LIST

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

UCSAPI_CALLBACK_PARAM_PTR_1 pCallback1
pCallback1.DataType = UCSAPI_CALLBACK_DATA_TYPE_USER_INFO

UCSAPI_GetUserDataFromTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_GetUserDataFromTerminal(

IN UCSAPI_INT32 ClientID,

IN UCSAPI_UINT32 TerminalID

IN UCSAPI_UINT32 UserID);

Description:

This API obtains the user data from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

UserID:

User ID

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_USER_DATA

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam :

```
UCSAPI_CALLBACK_PARAM_PTR_1  pCallback1;
pCallback1.DataType = UCSAPI_CALLBACK_DATA_TYPE_USER_DATA
```

3.3.3 Log related API

APIs to obtain log data stored at the terminal are described.

UCSAPI_GetAccessLogCountFromTerminal

Prototype:

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

Description:

This API obtains the number of authentication logs from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

LogType:

This parameter designates the log type to obtain. Refer to UCSAPI_GET_LOG_TYPE for available values.

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_PARAMETER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG_COUNT

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam :

UCSAPI_UINT32 nLogCount;

This parameter contains the number of logs.

UCSAPI_GetAccessLogFromTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_GetAccessLogFromTerminal(

IN UCSAPI_INT32 ClientID,

IN UCSAPI_INT32 TerminalID,

IN UCSAPI_GET_LOG_TYPE LogType);

Description:

This API obtains the authentication log data from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

LogType:

This parameter designates the log type to obtain. Refer to UCSAPI_GET_LOG_TYPE for available values.

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_PARAMETER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

In case several numbers of log records exist, the UCSAPI_CALLBACK_EVENT_GET_ACCESS_LOG event corresponding to the number of records are generated. Refer to the pCallback0->Progress value for the total number of logs and the index information of the current log.

lParam:

UCSAPI_CALLBACK_PARAM_PTR_1 pCallback1;

pCallback1->DataType = UCSAPI_CALLBACK_DATA_TYPE_ACCESS_LOG;

3.3.4 Authentication related API

APIs to implement authentication at the server are described.
In case terminal's authentication type is N/S, NO mode, the terminal requests authentication to the server.

UCSAPI_SendAuthInfoToTerminal

Prototype:

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

Description:

During 1:1 authentication, the terminal notifies the UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO event to the application program to obtain user's authentication information.
Then, the application program needs to include the user's authentication information in the UCSAPI_AUTH_INFO structure and send it to the terminal immediately.
The UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO event is always generated before the following events.
UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1,
UCSAPI_CALLBACK_EVENT_VERIFY_CARD,
CSAPI_CALLBACK_EVENT_VERIFY_PASSWORD

Parameters:

TerminalID:
Terminal ID

pUserAuthInfo:
The pointer of the structure that contains user's authentication information

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_VERIFY_USER_AUTH_INFO

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

UCSAPI_INPUT_ID_DATA_PTR pInputID;

The structure that contains the ID information entered from the terminal

UCSAPI_SendAuthResultToTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_SendAuthResultToTerminal(

IN UCSAPI_INT32 TerminalID,

IN UCSAPI_AUTH_RESULT_PTR pResult);

Description:

The terminal notifies the following events to an application program for user authentication.

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N (1:N fingerprint authentication request)

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1 (1:1 fingerprint authentication request)

UCSAPI_CALLBACK_EVENT_VERIFY_CARD (Card authentication request)

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD (Password authentication request)

Then, the application program needs to include the user's authentication result in the UCSAPI_AUTH_RESULT structure and send it to the terminal immediately.

Parameters:

TerminalID:

Terminal ID

pResult:

The pointer of the structure that contains authentication results

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_N

UCSAPI_CALLBACK_EVENT_VERIFY_FINGER_1_TO_1

UCSAPI_CALLBACK_EVENT_VERIFY_CARD

UCSAPI_CALLBACK_EVENT_VERIFY_PASSWORD

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam :

UCSAPI_INPUT_DATA_PTR pInputData;

The pointer of the structure that contains the sample data entered from the terminal

3.3.5 Terminal Management API

APIs to manage the terminal are described.

UCSAPI_GetTerminalCount

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_GetTerminalCount(
IN UCSAPI_UINT32* pTerminalCount);

Description:

This API obtains the number of terminals connected to the server.

Parameters:

pTerminalCount:

The pointer of the value to contain the number of terminals

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER

Callback:

N/A

Callback Parameters:

N/A

UCSAPI_GetFirmwareVersionFromTerminal

Prototype:

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

Description:

This API obtains the firmware version of the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_FW_VERSION

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

UCSAPI_DATA_PTR pVersion;

The pointer of the structure that contains version information

UCSAPI_UpgradeFirmwareToTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_UpgradeFirmwareToTerminal(

IN UCSAPI_UINT32 ClientID,

IN UCSAPI_UINT32 TerminalID,

IN UCSAPI_CHAR_PTR pFilePath);

Parameters:

This API upgrades the firmware of the designated terminal. Upgrade progress information is notified with the following event.

UCSAPI_CALLBACK_EVENT_FW_UPGRADING / UCSAPI_CALLBACK_EVENT_FW_UPGRADED

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

pFilePath:

The pointer of the value that contains the path of the firmware file

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_FW_UPGRADING

UCSAPI_CALLBACK_EVENT_FW_UPGRADED

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0 pCallback0;

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

lParam:

N/A

UCSAPI_GetOptionFromTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_GetOptionFromTerminal(

IN UCSAPI_UINT16 ClientID,

IN UCSAPI_UINT32 TerminalID);

Description:

This API obtains the option setting value from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_TERMINAL

Callback Parameters:

UCSAPI_CALLBACK_EVENT_GET_TERMINAL_OPTION

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_0 pCallback0;

lParam:

UCSAPI_TERMINAL_OPTION_PTR pOption;

UCSAPI_SetOptionToTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_SetOptionToTerminal(

IN UCSAPI_UINT32 ClientID,

IN UCSAPI_UINT32 TerminalID,

IN UCSAPI_TERMINAL_OPTION_PTR pOption);

Description:

This API sets up the option value of the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

pOption:

The pointer of the UCSAPI_TERMINAL_OPTION structure

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_POINTER

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_SET_TERMINAL_OPTION

wParam

UCSAPI_CALLBACK_PARAM_PTR_0 pCallBack0;

lParam:

N/A

UCSAPI_OpenDoorToTerminal

Prototype:

UCSAPI_RETURN UCSAPI UCSAPI_OpenDoorToTerminal(

IN UCSAPI_UINT32 ClientID,

IN UCSAPI_UINT32 TerminalID);

Description:

This API temporarily opens the locking device of the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Returns:

UCSAPIERR_NONE

UCSAPIERR_NOT_SERVER_ACTIVE

UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_OPEN_DOOR

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

N/A

UCSAPI_SetAccessControlDataToTerminal

Prototype:

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

Description:

This API sends the access control information to the designated terminal. Time zone, access time, holiday and access group information need to be sent separately. Access control information is used during authentication at the terminal. If no stored access control information is available, the terminal does not perform access control separately.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

pAccessControlData:

The pointer of the structure that contains access control information

Returns:

UCSAPIERR_NONE
UCSAPIERR_NOT_SERVER_ACTIVE
UCSAPIERR_INVALID_POINTER
UCSAPIERR_INVALID_TERMINAL

Callback:

UCSAPI_CALLBACK_EVENT_SET_ACCESS_CONTROL_DATA

Callback Parameters:

wParam:

UCSAPI_CALLBACK_PARAM_PTR_0 pCallback0;

lParam:

N/A

4. API Reference for COM

Properties and methods to use a COM module, UCSAPICOM.dll, are described in this chapter.

4.1 UCSAPI Object

As the main object to use UCSAPICOM.dll, it provides the basic functions of UCS SDK and operates as the basic object that obtains various child objects. To use SDK, this object must therefore be declared.

4.1.1 Properties

Various properties of UCSAPI objects are described.

ErrorCode

Prototype:

[ReadOnly] long ErrorCode;

Description:

This property contains the value on errors that occurred during executed method and property setup.

The value of 0 represents success, while all other values represent failure.

Errors that occurred during child object's method and property setup can also be obtained with this value.

ConnectionsOfTerminal

Prototype:

[ReadOnly] long ConnectionsOfTerminal;

Description:

This property contains the value of the number of terminals connected to the server.
This value can be obtained after the GetTerminalCount() method is called.

TerminalUserData

Prototype:

[ReadOnly] VARIANT TerminalUserData;

Description:

This property obtains the interface to obtain the user information from the terminal.
This interface needs to be obtained and used to obtain user information from EventGetUserInfoList / EvetGetUserData callback events after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called. Refer to IterminalUserData description for more details.

ServerUserData

Prototype:

[ReadOnly] VARIANT ServerUserData;

Description:

This property obtains the interface to send user information to the terminal.
This interface needs to be obtained and used to set up user information to be sent to the terminal before the AddUserToTerminal method is called. Refer to IServerUserData description for more details.

AccessLogData

Prototype:

[ReadOnly] VARIANT AccessLogData;

Description:

This property obtains the interface to obtain authentication log from the terminal.
This interface needs to be obtained and used to obtain authentication log from the EventGetAccessLog callback event occurred after the GetAccessLog method is called.

Refer to IAccessLogData description for more details.

AccessControlData

Prototype:

[ReadOnly] VARIANT AccessControlData;

Description:

This property obtains the interface to set up access control data with the terminal.
This interface needs to be obtained and used in order to set up the access rights data before the SetAccessControlDataToTerminal method is called. Refer to IAccessControlData description for more details.

4.1.2 Methods

Various methods of UCSAPI objects are described.

ServerStart

Prototype:

HRESULT ServerStart(long MaxTerminal, long Port);

Description:

This method initializes UCSAPI SDK and implements server functions.

Parameters:

MaxTerminal:

This parameter defines the maximum number of terminals that can be connected. SDK can improve speed by assigning internal memory capacity in advance according to the maximum number of terminals. If the number of connected terminals exceeds the maximum number, memory is increased automatically to increase the number of connections.

Port :

The communication port for terminal connection. The default value is 9870. If this value is

changed, the terminal's port value also needs to be changed.

Related Properties

ErrorCode

Callback Event:

EventTerminalConnected(long TerminalID, BSTR TerminalIP);

Event Parameters:

TerminalID:

Terminal ID

TerminalIP:

The character string that contains the terminal's IP address value

ServerStop

Prototype:

HRESULT ServerStop();

Description:

This method disconnects all connected terminals and stops server functions.

Parameters:

N/A

Related Properties

ErrorCode

Callback Event:

N/A

Event Parameters

N/A

GetTerminalCount

Prototype:

HRESULT GetTerminalCount(void);

Description:

This method obtains the number of terminals connected to the server. This number can be obtained using the ConnectionsOfTerminal property.

Parameters:

N/A

Related Properties

ErrorCode, ConnectionsOfTerminal

Callback Event:

N/A

Event Parameters

N/A

GetFirmwareVersionFromTerminal

Prototype:

HRESULT GetFirmwareVersionFromTerminal(long ClientID, long TerminalID);

Description:

This method obtains the firmware version of the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Related Properties

ErrorCode

Callback Event:

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

Event Parameters

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Version:

This parameter contains the firmware version value in the form of a character string.

UpgradeFirmwareToTerminal

Prototype:

HRESULT UpgradeFirmwareToTerminal (long ClientID, long TerminalID, BSTR FilePath);

Description:

This method upgrades the firmware of the designated terminal. Upgrade progress information is notified with the following event.

EventFirmwareUpgrading / EventFirmwareUpgraded

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

FilePath:

This parameter designates the full path of the firmware file with a character string.

Related Properties

ErrorCode

Callback Event:

EventFirmwareUpgrading(long ClientID, long TerminalID, long CurrentIndex, long TotalNumber);

EventFirmwareUpgraded(long ClientID, long TerminalID);

Event Parameters

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

CurrentIndex:

This parameter contains the index value of the data block under transmission.

TotalNumber:

This parameter contains the total number of data blocks to be sent.

OpenDoorToTerminal

Prototype:

HRESULE OpenDoorToTerminal(long ClientID, long TerminalID);

Description:

This method temporarily opens the locking device of the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Related Properties

ErrorCode

Callback Event:

EventOpenDoor (long ClientID, long TerminalID);

Event Parameters

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

4.2 IServerUserData Interface

The interface to send the user information to the terminal. To send user information to the terminal, the AddUserToTerminal method is called after setting up the user information related properties. After sending the user information, check whether transmission was completed normally by checking the error code of the EventAdduserToTerminal event.

4.2.1 Properties

Various properties of the IServerUserData interface are described.

UserID

Prototype:

[WriteOnly] long UserID;

Description:

This property designates the user ID value. This value can be designated only with numeric data of up to 8 digits.

UniqueID

Prototype:

[WriteOnly] BSTR UniqueID;

Description:

This property designates the unique ID (employee ID) with character string. This value can be used in place of UserID for user identification. The maximum data size that can be designated is 20bytes.

UserName

Prototype:

[WriteOnly] BSTR UserName;

Description:

This property designates the user name value with character string. The maximum data size that can be designated is 16bytes.

AccessGroup

Prototype:

[WriteOnly] BSTR AccessGroup;

Description:

This property designates the access group code value with character string. The code value consists of a 4byte character string.

SecurityLevel

Prototype:

[WriteOnly] long SecurityLevel;

Description:

This property can set up the authentication security level to be used in fingerprint authentication. It can have any of the following values.

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

The default level is 4 for 1:1 authentication and 5 for 1:N authentication.

IsCheckSimilarFinger

Prototype:

[WriteOnly] BOOL IsCheckSimilarFinger;

Description:

When adding user's fingerprint data to the terminal, the process of whether to check for

a similar fingerprint or not is designated.

If this value is designated as true, the terminal checks if a similar fingerprint exists by comparing with the fingerprints of all registered users. If a similar fingerprint is detected, registration fails. Since this flag can slow down user addition job by the terminal, the performance may be degraded if there are a large number of registered users.

IsAdmin

Prototype:

[WriteOnly] BOOL IsAdmin;

Description:

This property can designate a user as an administrator. For the terminal with more than 1 registered administrator, the use of the terminal menu can be restricted through the administrator logon process during entry to the setup menu.

IsIdentify

Prototype:

[WriteOnly] BOOL IsIdentify;

Description:

This property can designate to allow the user to use 1:N fingerprint authentication.

Password

Prototype:

[WriteOnly] BSTR Password;

Description:

In case that the user uses the password authentication method, this property designates a password character string. The maximum size of data that can be designated is 8bytes.

4.2.2 Methods

Various methods of the IServerUserData interface are described.

SetAuthType

Prototype:

HRESULT SetAuthType(BOOL AndOperation, BOOL Finger, BOOL FPCard, BOOL Password, BOOL Card, BOOL CardID);

Description:

This method designates user's authentication type. Each authentication type can be used through AND or OR combination according to the AndOperation flag.

Parameters:

AndOperation:

This parameter designates to allow the use of each authentication type through AND or OR combination.

1 is set for AND combination and 0 for OR combination. For more details, refer to User Property in Section 1.6 Terminology Description.

Finger:

This parameter designates to allow the use of fingerprint authentication.

FPCard:

This parameter designates to allow the use of fingerprint card authentication. The fingerprint card uses a method that authenticates by storing the fingerprint information at the smart card.

Password:

This parameter designates to allow the use of password authentication.

Card:

This parameter designates to allow the use of card authentication.

CardID:

This parameter designates to allow the use of RFID as UserID or UniqueID. CardID does not use the card's RFID as authentication tool, but instead, the card's RFID is simply used as an identifier like UserID.

It must be designated using AND combination with other authentication type.

Related methods

AddUserToTerminal, SendAuthInfoToTerminal

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetFPSampleData

Prototype:

HRESULT SetFPSampleData(BOOL bInitialize, long nSrcFPDataType, long nFPDataSize, VARIANT FPData1, VARIANT FPData2);

Description:

This method designates the binary stream data of the template for each finger of FIR converted for fingerprint authentication. For more detailed description on the template, refer to UCBioBSP SDK.

Parameters:

bInitialize:

This parameter designates whether to initialize FIR data and produce new data or not.
If this value is false, the added template data are appended to the FIR data produced internally to produce one FIR data with several template data. If this value is true, all existing FIR data are deleted and new data are produced.

nSrcFPDataType :

The type information of the template to be added. Refer to UCBioAPI_TEMPLATE_TYPE for relevant values.

nFPDataSize :

The size data of the template to be added

FPData1:

Template data to be added. (Binary stream data)

FPData2:

The second template data of a finger to be added. (Binary stream data)

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetCardData

Prototype:

HRESULT SetCardData(BOOL bInitialize, BSTR RFID);

Description:

This method designates RFID data for card authentication.

Parameters:

bInitialize:

This parameter designates whether to initialize RFID data and produce new data or not.
If this value is false, the added RFID data are appended to the RFID data produced internally to produce several RFID data.
If this value is true, all existing RFID data are deleted and new data are produced.

RFID :

This parameter designates the RFID value to be added with the character string. The maximum size of data that can be designated is 16bytes.

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetPictureData

Prototype:

HRESULT SetPictureData(long PictureDataLength, BSTR PictureDataType, VARIANT PictureData);

Description:

This method designates the picture data with binary stream.

Parameters:

PictureDataLength:

This parameter designates the length of picture data.

PictureDataType:

This parameter designates the type value of picture data with the character string. (Currently, only "JPG" is supported.)
It designates the file extension value with the character string.

PictureData:

Picture data to be added. (Binary stream data)

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetAccessDate

Prototype:

HRESULT SetAccessDate(long AccessDateType, long StartYear, long StartMonth, long StartDay, long EndYear, long EndMonth, long EndDay);

Description:

This method designates the period allowed for access or the period not allowed for access. Three access period data can be designated; not used, allowed access period, and access restriction period. The following values are available.

Parameters:

AccessDateType:

This parameter designates the type of access period data. The following values are available.

0 - Not used
1 - Period allowed for authentication designated
2 - Period not allowed for authentication designated

StartYear / StartMonth / StartDay:

This parameter designates the start date of the access period.

EndYear / EndMonth / EndDay:

This parameter designates the end date of the access period.

Related properties:

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

AddUserToTerminal

Prototype:

HRESULT AddUserToTerminal(long ClientID, long TerminalID, BOOL IsOverwrite);

Description:

This method sends the user information to the designated terminal. User information needs to be produced using the user information related properties and methods before the AddUserTerminal method is called.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

IsOverwrite:

This parameter designates whether to overwrite an already registered user or not. The default value is 1.

Related properties:

ErrorCode

Callback Event:

EventAddUser

Event Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

※ Note

In case of sending multiple numbers of user data to the terminal using the AddUserToTerminal method, the EventAddUser callback event must be checked after the AddUserToTerminal method is called. The next user data is sent only after the transmission is processed normally.

4.3 ITerminalUserData Interface

The interface related to terminal's user information management and get function. This interface needs to be obtained and used in order to obtain or delete the number of users, user information list, and user data.

4.3.1 Properties

Various properties of the ITerminalUserData interface are described.

CurrentIndex / TotalNumber

Prototype:

```
[ReadOnly]    long    CurrentIndex;
[ReadOnly]    long    TotalNumber;
```

Description:

In case of obtaining multiple numbers of user information lists, this property contains the total number of lists and the index of the current record. It can be obtained after the GetUserInfoListFromTerminal method is called.

Related methods:

GetUserInfoListFromTerminal,

UserID

Prototype:

```
[ReadOnly]    long    UserID;
```

Description:

This property contains the user ID value.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

UniqueID

Prototype:

[ReadOnly] BSTR UniqueID;

Description:

This property contains the unique ID value (employee ID).

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

UserName

Prototype:

[ReadOnly] BSTR UserName;

Description:

This property contains the user name value.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

AccessGroup

Prototype:

[ReadOnly] BSTR AccessGroup;

Description:

This property contains the access group code value.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

IsAdmin

Prototype:

[ReadOnly] BOOL IsAdmin;

Description:

This property contains the flag value on whether the user is an administrator or not.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

IsIdentify

Prototype:

[ReadOnly] BOOL IsIdentify;

Description:

This property contains the flag value on whether 1:N fingerprint authentication is allowed or not.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber, FPSampleData, SecurityLevel

AccessDateType

Prototype:

[ReadOnly] long AccessDateType;

Description:

This property contains the type value of access period data. The following values are available.

It can be obtained after the GetUserDataFromTerminal method is called.

0 - Not used
1 - Period allowed for authentication designated
2 - Period not allowed for authentication designated

Related methods:

GetUserDataFromTerminal

Related properties:

StartAccessDate, EndAccessDate

StartAccessDate/EndAccessDate

Prototype:

[ReadOnly] BSTR StartAccessDate;

[ReadOnly] BSTR EndAccessDate;

Description:

This property contains the start/end date of access period.

The data format is yyyy-MM-dd.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserDataFromTerminal

Related Properties:

AccessDateType

SecurityLevel

Prototype:

[ReadOnly] long SecurityLevel;

Description:

This property contains the authentication security level to be used in fingerprint authentication.

It can be obtained after the GetUserDataFromTerminal method is called.

The following values are available.

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

The default value is 4 for 1:1 authentication and 5 for 1:N authentication.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber, FPSampleData

IsAndOperation

Prototype:

[ReadOnly] BOOL IsAndOperation;

Description:

This property contains the flag value on the AND/OR combination of the authentication type. This value allows the use of fingerprint, card, and password authentication types through AND or OR combination. For more details, refer to User Property in Section 1.6 Terminology Description.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, IsFPCard, IsCard, IsPassword

IsFinger

Prototype:

[ReadOnly] BOOL IsFinger;

Description:

This property contains the flag value that allows user's fingerprint authentication.
It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber, FPSampleData, SecurityLevel

IsFPCard

Prototype:

[ReadOnly] BOOL IsFPCard;

Description:

This property contains the flag value that allows user's fingerprint card authentication.
It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation

IsCard

Prototype:

[ReadOnly] BOOL IsCard;

Description:

This property contains the flag value that allows user's card authentication.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, CardNumber, RFID

IsCardID

Prototype:

[ReadOnly] BOOL IsCardID;

Description:

This property contains the flag value on whether Card user's RFID is used as ID for user identification or not.

CardID does not use the card's RFID as authentication tool, but instead, the card's RFID is simply used as an identifier like UserID. It must be designated using AND combination with other authentication type.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation

IsPassword

Prototype:

[ReadOnly] BOOL IsPassword;

Description:

This property contains the flag value that allows password authentication.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserInfoListFromTerminal, GetUserDataFromTerminal

Related properties:

IsAndOperation, Password

Password

Prototype:

[ReadOnly] BSTR Password;

Description:

This property contains the password value in the form of a character string. The maximum size of data that can be designated is 8bytes.

It can be obtained after the GetUserDataFromTerminal method is called.

Related methods:

GetUserDataFromTerminal

Related properties:

IsPassword

CardNumber

Prototype:

[ReadOnly] long CardNumber;

Description:

In case that the user uses the card authentication method, this property contains the value of the number of registered RFIDs.

It can be obtained after the GetUserDataFromTerminal method is called.

Related methods:

GetUserDataFromTerminal

Related properties:

IsCard, RFID

RFID

Prototype:

[ReadOnly] BSTR RFID(long nIndex);

Description:

In case that the user uses the card authentication method, this property contains the data value of registered RFID.

It can be obtained after the GetUserDataFromTerminal method is called.

Parameters:

nIndex

The index number of RFID to be obtained

Related methods:

GetUserDataFromTerminal

Related properties:

IsCard, CardNumber

PictureDataLength

Prototype:

[ReadOnly] long PictureDataLength;

Description:

This property contains the size value of picture data.

It can be obtained after the GetUserDataFromTerminal method is called.

Related methods:

GetUserDataFromTerminal

Related properties:

PictureData

PictureData

Prototype:

[ReadOnly] VARIANT PictureData;

Description:

This property contains the picture data value in the form of a binary stream. The length value of data is contained in the PictureDataLength property.

It can be obtained after GetUserInfoListFromTerminal / GetUserDataFromTerminal methods are called.

Related methods:

GetUserDataFromTerminal

Related properties:

PictureDataLength

TotalFingerCount

Prototype:

[ReadOnly] long TotalFingerCount;

Description:

This property contains the total number of fingers of converted FIR.
It can be obtained after the GetUserDataFromTerminal method is called.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, FingerID, FPSampleDataLength, SampleNumber, FPSampleData

FingerID

Prototype:

[ReadOnly] long FingerID(long nIndex);

Description:

This property contains the finger ID information of converted FIR in the form of an array.
nIndex can have a value between 0 and (TotalFingerCount-1).
It can be obtained after the GetUserDataFromTerminal method is called.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger,TotalFingerCount, FPSampleDataLength, SampleNumber, FPSampleData

SampleNumber

Prototype:

[ReadOnly] long SampleNumber;

Description:

This property contains the number of templates for each finger of converted FIR. It contains the value of 1 or 2.
It can be obtained after the GetUserDataFromTerminal method is called.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, FPSampleData

FPSampleData

Prototype:

[ReadOnly] VARIANT FPSampleData(long nFigerID, long nSampleNum);

Description:

This property contains template's binary stream data for each finger of converted FIR.

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

The length value of binary stream data can be obtained using the FPSampleDataLength property.

It can be obtained after the GetUserDataFromTerminal method is called.

Parameters:

nFingerID:

Finger ID number to be obtained

nSampleNum:

Sample number to be obtained. The value of 0 or 1 is used.

Related methods:

GetUserDataFromTerminal

Related properties:

IsFinger, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber

4.3.2 Methods

Various methods of the ITerminalUserData interface are described.

GetUserCountFromTerminal

Prototype:

HRESULT GetUserCountFromTerminal(long ClientID, long TerminalID);

Description:

This method obtains the total number of registered users from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Related Properties

ErrorCode

Callback Event:

EventGetUserCount

Event Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

AdminNumber:

The number of registered administrators

UserNumber:

The number of registered users

GetUserDataFromTerminal

Prototype:

HRESULT GetUserDataFromTerminal(long ClientID, long TerminalID, long UserID);

Description:

This method obtains the user data from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

UserID:

User ID

Related Properties

ErrorCode, UserID, UserName, UniqueID, AccessGroup, AccessDateType, StartAccessDate, EndAccessDate, IsAdmin, IsIdentify, IsAndOperation, IsFinger, IsFPCard, IsCard, IsPassword, IsCardID, Password, CardNumber, RFID, SecurityLevel, TotalFingerCount, FingerID, FPSampleDataLength, SampleNumber, FPSampleData, PictureDataLength, PictureData

Callback Event:

EventGetUserData

Event Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

DeleteUserFromTerminal

Prototype:

HRESULT DeleteUserFromTerminal(long ClientID, long TerminalID, long UserID);

Description:

This method deletes the user data from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

UserID:

User ID

Related Properties

ErrorCode

Callback Event:

EventDeleteUser

Event Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

UserID:

User ID

DeleteAllUserFromTerminal

Prototype:

HRESULT DeleteAllUserFromTerminal(long ClientID, long TerminalID);

Description:

This method deletes all user data from the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

Related Properties

ErrorCode

Callback Event:

EventDeleteAllUser

Event Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

4.4 IAccessLogData Interface

The interface related to the authentication log get function of the terminal. This interface is required to obtain the authentication log of the terminal.

4.4.1 Properties

Various properties of the IAccessLogData interface are described.

CurrentIndex / TotalNumber

Prototype:

```
[ReadOnly]    long    CurrentIndex;
[ReadOnly]    long    TotalNumber;
```

Description:

In case of obtaining several numbers of log records, this property contains the total number of records and the index of the current record.

It can be obtained after GetAccessLogCountFromTerminal/GetAccessLogFromTerminal methods are called.

Related methods:

GetAccessLogFromTerminal

UserID

Prototype:

```
[ReadOnly]    long    UserID;
```

Description:

This property contains the user ID value.

It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

AuthType

Prototype:

[ReadOnly] long AuthType;

Description:

This property contains the authentication type value. The following values are available.

0 - 1:N fingerprint authenticaiton
1 - 1:1 fingerprint authenticaiton
2 - Fingerprint and card authentication
3 - Card authentication
4 - Password authentication

It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

AuthMode

Prototype:

[ReadOnly] long AuthMode;

Description:

This property contains the authentication mode value. The following values are available.

0 - Office start
1 - Office leave
2 - General (General access)
3 - Work outside
4 - Return to office

It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

DateTime

Prototype:

[ReadOnly] BSTR DateTime;

Description:

This property contains the authentication time data in the form of a character string.
The data format is "yyyy-MM-dd hh:mm:ss".
It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

IsAuthorized

Prototype:

[ReadOnly] BOOL IsAuthorized;

Description:

This property contains the authentication result value. This value is 0 for authentication success and 1 for failure.
It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

RFID

Prototype:

[ReadOnly] BSTR RFID;

Description:

In case the authentication type is card, this property contains the RFID value in the form of a character string.
It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

PictureDataLength

Prototype:

[ReadOnly] long PictureDataLength;

Description:

This property contains the size value of picture data.

It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

Related properties:

PictureData

PictureData

Prototype:

[ReadOnly] VARIANT PictureData;

Description:

This property contains picture data in the form of a binary stream. The PictureDataLength property contains the length value of data.

It can be obtained after the GetAccessLogFromTerminal method is called.

Related methods:

GetAccessLogFromTerminal

Related properties:

PictureDataLength

4.4.2 Methods

Various methods of the IAccessLogData interface are described.

GetAccessLogCountFromTerminal

Prototype:

HRESULT GetAccessLogCountFromTerminal(long ClientID, long TerminalID, long LogType);

Description:

This method obtains the number of authentication logs stored at the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

LogType:

This parameter designates the type of the log to obtain. The following values are available.

0 - New log
1 - Log already sent to the server
2 - All stored logs

Related properties

ErrorCode

Callback Event:

EventGetAccessLogCount

Event Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

LogCount:

The number of authentication logs

GetAccessLogFromTerminal

Prototype:

HRESULT GetAccessLogFromTerminal(long ClientID, long TerminalID, long LogType);

Description:

This method obtains the authentication log stored at the designated terminal.

Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

LogType:

This parameter designates the type the log to obtain. The following values are available.

0 - New log
1 - Log already sent to the server
2 - All stored logs

Related Properties

ErrorCode, TotalNumber, CurrentIndex, UserID, DateTime, AuthType, AuthMode, IsAuthorized, RFID, PictureDataLength, PictureData

Callback Event:

EventGetAccessLog

Event Parameters:

ClientID:

ID of the client that requested a job. (It is used in client/server model development.)

TerminalID:

Terminal ID

4.5 IAccessControlData Interface

The interface to transmit access control data to the terminal. This interface needs to be loaded and used to set up access control information with the terminal.

4.5.1 Properties

Various properties of IAccessControlData interface are described.

4.5.2 Methods

Various methods of IAccessControlData interface are described.

InitData

Prototype:

HRESULT InitData(void);

Description:

This method designates whether to initialize access control data and create new data or not.

Parameters:

N/A

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetTimeZone

Prototype:

HRESULT SetTimeZone(BSTR Code, long nIndex, long StartHour, long StartMinute, long EndHour, long EndMinute);

Description:

This method adds a time zone during a day allowed for authentication.
Up to 128 time zone codes can be added, and up to 12 time zones can be added to a single time zone code. SDK contains the added information in the form of an array.

Parameters:

Code:

As the identification code value of the time zone to be set up, this parameter is a fixed 4byte character string.

nIndex:

Index on time zone information. This value can have a value between 0 and 11.

StartHour / StartMinute:

This parameter designates the start time of the time zone.

EndHour / EndMinute:

This parameter designates the end time of the time zone.

Related methods

SetAccessControlDataToTerminal

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetAccessTime

Prototype:

HRESULT SetAccessTime(BSTR Code, BSTR Sun, BSTR Mon, BSTR Tue, BSTR Wed, BSTR Thu, BSTR, Fri, BSTR Sat, BSTR Hol, BSTR Holiday);

Description:

This method adds an allowed access time for each day of the week.
Up to 128 allowed access time codes can be added, and SDK contains the added information in the form of an array.

Parameters:

Code:

As the identification code value of the allowed access time to be set up, this parameter is a fixed 4byte character string.

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

These parameters designate the allowed access time zone code for each day of the week to be used during authentication and the allowed access time zone code to be applied to holidays.

Holiday:

This parameter designates the holiday code that was set up at the SetHoliday method. The time zone of the Hol code is applied to the designated holiday code.

Related methods

SetAccessControlDataToTerminal

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetHoliday

Prototype:

HRESULT SetHoliday(BSTR Code, long nIndex, long Month, long Day);

Description:

This method adds holiday information.

Up to 64 holiday codes can be added, and up to 32 holidays can be added to a single holiday code. SDK contains added information in the form of an array.

Parameters:

Code:

As the identification code value of the holiday to be set up, this parameter is a fixed 4byte character string.

nIndex:

Index value of the holiday to be added. This value can have a number between 0 and 63.

Month / Day:

This parameter designates the date for the holiday.

Related methods

SetAccessControlDataToTerminal

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetAccessGroup

Prototype:

HRESULT SetAccessGroup(BSTR Code, long nIndex, BSTR AccessTime);

Description:

This method adds access group information.

Up to 128 access group codes can be added, and up to 4 allowed access time codes can be added to a single access group code. SDK contains added information in the form of an array.

Parameters:

Code:

As the identification code value of the access group to be set up, this parameter is a fixed 4byte character string.

nIndex:

The index value of the allowed access time code to be added. This value can have a number between 0 and 3.

AccessTime:

This parameter designates the allowed access time code to be used at the access group.

Related methods

SetAccessControlDataToTerminal

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SetAccessControlDataToTerminal

Prototype:

HRESULT SetAccessControlDataToTerminal(long ClientID, long TerminalID, long DataType);

Description:

This method sends to the designated terminal the access control data added by SetTimeZone, SetAccessTime, SetHoliday, SetAccessTime and SetAccessGroup methods. Time zone, access time, holiday and access group information need to be sent separately. Access control information is used at the terminal during authentication. In case stored access control information is not available, the terminal does not implement access control.

Parameters:

ClientID:

ID of the client that requested a job (It is used in client/server model development.)

TerminalID:

Terminal ID

DataType:

This parameter designates access control data to be sent to the terminal. The following values are available.

0 - Time zone information
1 - Holiday information
2 - Allowed access time information
3 - Access group information

Related methods

SetTimeZone, SetHoliday, SetAccessTime, SetAccessGroup

Related properties

ErrorCode

Callback Event:

HRESULT EventSetAccessControlData(long ClientID, long TerminalID, long DataType);

Event Parameters:

ClientID:

ID of the client that requested a job (It is used in client/server model development.)

TerminalID:

Terminal ID

DataType:

The type value of access control data

4.6 IServerAuthentication Interface

The interface to implement server authentication. The application program requires this interface to reply to the user authentication request from the terminal.

4.6.1 Properties

Various properties of the IServerAuthentication interface are described.

4.6.2 Methods

Various methods of the IServerAuthentication interface are described.

SetAuthType

Prototype:

HRESULT SetAuthType(BOOL AndOperation, BOOL Finger, BOOL FPCard, BOOL Password, BOOL Card, BOOL CardID);

Description:

This method designates the user's authentication type. Each authentication type can be used through AND or OR combination according to the AndOperation flag. It must be used before the SendAuthInfoToTerminal method is called.

Parameters:

AndOperation:

If this value is true, authentication with AND combination is allowed. If this value is false, authentication with OR combination is allowed.

Finger:

This parameter designates to allow the use of fingerprint authentication.

FPCard:

This parameter designates to allow the use of fingerprint card authentication. The fingerprint card uses the method that authenticates by storing the fingerprint information at the smart card.

Password:

This parameter designates to allow the use of password authentication.

Card:

This parameter designates to allow the use of card authentication.

CardID:

This parameter designates to allow the use of RFID as UserID or UniqueID. CardID does not use card's RFID as authentication tool, but instead, the card's RFID is simply used as an identifier like UserID.

It must be designated using AND combination with other authentication type.

Related methods

SendAuthInfoToTerminal

Related properties

ErrorCode

Callback Event:

N/A

Event Parameters:

N/A

SendAuthInfoToTerminal

Prototype:

HRESULT SendAuthInfoToTerminal(long TerminalID, long UserID, BOOL IsAccessibility, long ErrorCode);

Description:

When the terminal operates in server authentication mode, the application program needs to immediately send user's authentication information to the terminal in response to the EventAuthTypeWithUserID/EventAuthTypeWithUniqueID event. It must be used after the SetAuthType method is called.

Parameters:

TerminalID:

Terminal ID

UserID:

ID of the user who attempted authentication

IsAccessibility:

This parameter designates the flag value on whether the user has access rights or not. If this value is false, authentication fails at the terminal.

Related Properties

ErrorCode

Callback Event:

N/A

Event Parameters

N/A

SendAuthResultToTerminal

Prototype:

HRESULT SendAuthResultToTerminal(long TerminalID, long UserID, BOOL IsAccessibility, BOOL IsVisitor, BOOL IsAuthorized, BSTR AuthorizedTime, long ErrorCode);

Description:

For user authentication, the terminal notifies the following events to the application program.

EventVerifyCard, EventVerifyPassword, EventVerifyFinger_1_TO_1, EventVerifyFinger_1_TO_N Then, the application program must send the user's authentication result to the terminal immediately.

Parameters:

TerminalID:

Terminal ID

UserID:

ID of the authenticated user or the user who attempted authentication

IsAccessibility:

This parameter designates the flag on whether the user has access rights or not. If this value is false, authentication fails at the terminal.

IsVistor:

This parameter designates whether the user is a visitor or not.

IsAuthorized:

This parameter designates whether authentication is success or not.

AuthorizedTime:

This parameter designates the authentication time. The character string in the form of "yyyy-MM-dd hh:mm:ss" is designated for this value.

ErrorCode:

This parameter designates the code of the error that occurs during authentication. The value of 0 represents no error. Refer to the ErrorCode table for more information.

Related Properties

ErrorCode

Callback Event:

N/A

Event Parameters

N/A