IC Card プロバイダ

デンソー 非接触 IC カードリーダライタ

Version 1.0.2

ユーザーズ ガイド

July 9, 2020

【備考】

【改版履歴】

バージョン	日付	内容
1.0.0.0	2010-09-07	初版
1.0.1.0	2010-10-28	システム変数“@LastReceive”, “@SendReceive”追加
1.0.1	2012-07-17	ドキュメントのバージョンルールを変更
	2017-12-14	対応機種追加
1.0.2	2020-07-09	機器と未接続状態時に AddController した際の挙動を改善 GetReaderWriterInfo と GetCardCommCtrlInfo で製造情報を取得できるように修正

【対応機器】

機種	バージョン	注意事項
PR-450		
PR-550		
QK12-IC		
PR-700		

【動作確認機器】

機種	バージョン	注意事項
PR-450UDM		
PR-700UDM		

目次

1. はじめに

本書は，デンソーウェーブ製 非接触 IC カードリーダライタ（以下，リーダライタ）用 ORiN プロバイダである IC Card プロバイダのユーザーズガイドです．

IC Card プロバイダは，リーダライタの通信コマンド使用して，IC カードとの通信を行います．

本書は，この IC Card プロバイダの機能と実装されているメソッドについて説明します．

1.1. 接続構成

図 1-1 が本プロバイダとリーダライタの全体構成図になります．

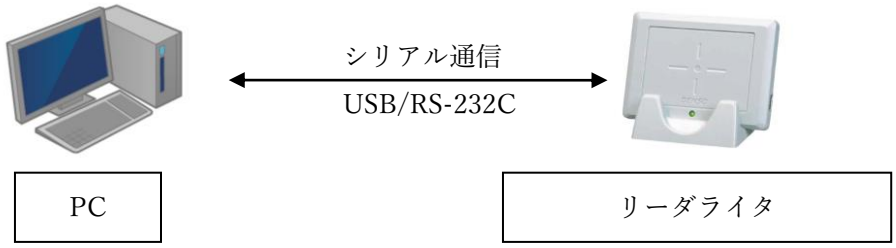

図 1-1 接続構成図

リーダライタとプロバイダはシリアル通信を行います．通信ケーブルには RS232C ケーブルと USB ケーブルが存在し，リーダライタの種類によって対応しているケーブルが異なります．また，USB ケーブルをお使いの場合は別途仮想 COM ドライバの Active USB-COM ポートドライバや，IC カード用 USB デバイスドライバ[1]が必要になります．

接続方式		必要ソフト
RS-232C		不要
USB	PR-700 シリーズ	Active USB-COM ポートドライバ
	PR-450 シリーズ	IC カード用 USB デバイスドライバ

1.1.1. Active USB-COM ポートドライバによる USB 接続

Active USB-COM ポートドライバのインストーラに従ってインストール後，リーダライタを接続します．

デバイスマネージャにてポート(図 1-2 を参照)を確認します．Connected と書かれている COM ポート番号が本プロバイダで接続する際に必要となるので控えておいてください．

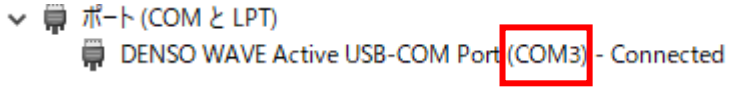

図 1-2 Active USB-COM ポートドライバによる接続

[1] 各種ドライバは DENSO-WAVE の HP(https://www.denso-wave.com/ja/)より入手できます．

1.1.2. IC カード用 USB デバイスドライバによる USB 接続

IC カード用 USB デバイスドライバのインストーラに従ってインストール後，リーダライタを接続します．

デバイスマネージャにてポート(図 1-3 を参照)を確認します．DENSO WAVE USB-SER と書かれている COM ポート番号が本プロバイダで接続する際に必要となるので控えておいてください．

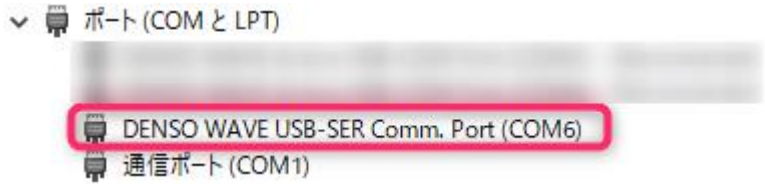

図 1-3 IC カード用 USB デバイスドライバによる接続

2. プロバイダの概要

2.1. 概要

IC Card プロバイダは, リーダライタの各コマンドを実行することで IC カードへのアクセスを実行します.

リーダライタのリーダライタコマンドは CaoController, カード通信コマンドは CaoExtension で実装しています.

IC Card プロバイダで対応している IC カードは Mifare のみです. また, リーダライタの暗号化通信には対応していません.

IC Card プロバイダのファイル形式は DLL(Dynamic Link Library)となっており, その詳細は表 2-1 のようになっています.

表 2-1 ICCard プロバイダ

ファイル名	CaoProvICCard.dll
ProgID	CaoProv.DENSO.ICCard
レジストリ登録[2]	regsvr32 CaoProvICCard.dll
レジストリ登録の抹消	regsvr32 /u CaoProvICCard.dll

[2] ORiN SDK でインストールした場合は手動で登録／抹消する必要はありません.

2.2. メソッド・プロパティ

2.2.1. CaoWorkspace::AddController メソッド

AddController では，リーダライタとの通信を行う CaoController オブジェクトを作成します．

このメソッドではオプション文字列を使用して，リーダライタとの通信設定を指定することができます．

以下に AddController の引数仕様を示します．

書式 AddController(<bstrCtrlName:BSTR>,<bstrProvName:BSTR>,
<bstrPcName:BSTR > [,<bstrOption:BSTR>])

bstrCtrlName : [in] コントローラ名

bstrProvName : [in] プロバイダ名．固定値 =”CaoProv.DENSO.ICCard”

bstrPcName : [in] プロバイダの実行マシン名

bstrOption : [in] オプション文字列

表 2-2 CaoWorkspace::AddController のオプション文字列

オプション	説明
Port [=<COM Port>]	COM ポート番号 （デフォルト：1）
BaudRate [=<BaudRate>]	通信速度 （デフォルト：9600）
Timeout [=<タイムアウト時間>]	送受信時のタイムアウト時間．(ミリ秒) （デフォルト：500）

2.2.2. CaoController::AddExtension メソッド

AddExtension では，IC カードと通信を行う CaoExtension オブジェクトを作成します．

書式 AddExtension(<bstrName:BSTR > [,<bstrOption:BSTR>])

bstrName : [in] コマンド名．固定値 =”@Mifare”

bstrOption : [in] オプション文字列(未使用)

2.2.3. CaoController::AddVariable メソッド

変数オブジェクトを作成します．指定できる変数名は 2.3.1 を参照して下さい．

書式 AddVariable(<bstrName:BSTR > [,<bstrOption:BSTR>])

bstrName : [in] 変数名

bstrOption : [in] オプション文字列(未使用)

2.2.4. CaoController::Execute メソッド

リーダライタのリーダライタコマンドを実行します.

使用できるコマンド名と詳細は 3.3 を参考にしてください.

2.2.5. CaoController::get_VariableNames プロパティ

変数名の一覧を取得します.

2.2.6. CaoExtension::Execute メソッド

リーダライタのカード通信コマンドを実行します.

使用できるコマンド名と詳細は 3.4 を参考にしてください.

2.2.7. CaoVariable::get_Value プロパティ

変数に対応した値を取得します. 取得する値の詳細については, 2.3.1 を参照してください.

2.3. 変数一覧

2.3.1. コントローラクラス

コントローラクラスで実装されている変数の一覧を示します.

表 2-3 コントローラクラス システム変数一覧

変数名	データ型	説明	属性	
			get	put
@LastReceive	VT_UI1 | VT_ARRAY	最後に受信したパケット.	○	×
@LastSend	VT_UI1 | VT_ARRAY	最後に送信したパケット	○	×

2.4. エラーコード

IC Card プロバイダで定義されている独自のエラーコードはありません.

ORiN2 共通エラーについては,「ORiN2 プログラミングガイド」のエラーコードの章を参照してください.

3. コマンドリファレンス

3.1. コマンド一覧

本プロバイダは Controller クラスと Extension クラスでそれぞれ Execute コマンドが実装されています. Controller クラスのコマンドはリーダライタ自体に関する制御, 設定を行います. Extension クラスのコマンドはカード通信に関する制御, 設定を行います.
以下に各クラスで使用できるコマンドの一覧を示します.

表 3-1 コントローラクラスのコマンド一覧

コマンド	機能	
ResetMain	リーダライタを初期化します.	P. 12
CheckCtrlState	リーダライタ内部の各制御部が正常に起動しているか確認します.	P. 12
GetReaderWriterInfo	リーダライタの固有情報を取得します.	P. 13
ModifyHostBaudRate	リーダライタと上位機器間の通信速度を変更します.	P. 13
SetLED	リーダライタの LED を制御します.	P. 14

表 3-2 拡張ボードクラスのコマンド一覧

コマンド	機能	
GetCardCommCtrlInfo	カード通信制御部の情報を取得します.	P. 14
SetCarrierState	カードへの電力(キャリア)を供給し, 通信可能な状態にします. また, 電力(キャリア)を停止し, 通信を終了します.	P. 15
Authentication1	リーダライタに対し, 認証用乱数1(Rnd1)を要求します. (Authentication2 実行準備)	P. 15
Authentication2	上位機器とリーダライタの相互認証を実施します.	P. 15
Request	リーダライタのカード検知範囲内にあるカードの有無を確認します.	P. 16
Anticollision	カードに対し, アンチコリジョン処理を行い, カードのシリアル No.を持つカードを選択します.	P. 16
Select	指定したシリアル No.を持つカードを選択します.	P. 16
Authentication	カードの相互認証処理を行います. 相互認証キーは直接指定します.	P. 17
AuthenticationInMemory	カードの相互認証処理を行います. 相互認証キーは, Mifare 専用メモリから選択します.	P. 17
Read	指定したアドレスのデータを読み出します.	P. 18
Write	指定したアドレスへデータを書き込みます.	P. 18
DecrementTrans	指定したアドレスのデータに対し, 指定した値分の減算処理を実行し, その結果を指定したアドレスに転送します.	P. 18

IncrementTrans	指定したアドレスのデータに対し，指定した値分の加算処理を実行し，その結果を指定したアドレスに転送します.	P. 19
RestoreTrans	指定したアドレスのデータに対し，リストア処理を実行し，その結果を指定したアドレスに転送します.	P. 19
Transfer	カードに対する転送処理を実施します.	P. 20
Halt	カードをホルト状態にします.	P. 20
AutoSelect	カードに対する Requesut 処理から Select 処理までを一括して行います.	P. 22
AutoLogin	カードに対する Request 処理から Authentication 処理までを一括して行います. 相互認証キーは直接指定します.	P. 22
AutoLoginInMemory	カードに対する Request 処理から Authentication 処理までを一括して行います. 相互認証キーは Mifare 専用メモリから選択します.	P. 23
DecrementRead	指定したアドレスのデータに対し，Decrement 処理を実行後，転送先に指定したアドレスのデータを読み出します.	P. 23
IncrementRead	指定したアドレスのデータに対し，Increment 処理を実行後，転送先に指定したアドレスのデータを読み出します.	P. 24
RestoreRead	指定したアドレスのデータに対し，Restore 処理を実行後，転送先に指定したアドレスのデータを読み出します.	P. 24
WriteRead	指定したアドレスのデータに対し，Write 処理を実行後，データを読み出します.	P. 25
ValueWrite	指定したアドレスに Value データを書き込み後，Value データを読み取ります.	P. 25
LoadKey	Mifare カード相互認証要キーをリーダライタに登録します.	P. 25
AutoRead	特定エリアの情報を読取ります.	

3.2. Execute コマンドによるカード通信

本プロバイダを使ってカード通信を行うには，以下の手順に従ってください.

なお，カード通信には認証キーが必要ですので，予め準備ください.

手順	コマンド
1	CaoController::ResetMain リーダライタの初期化します.
2	CaoController::AddExtension カード通信のための Extension クラスを作成します.
3	CaoExtension::SetCarrierState キャリア出力を ON にします.
4	CaoExtension::Authentication1 認証用乱数 1 を取得します.
5	CaoExtension::Authentication2 手順 4 で取得した乱数 1 を引数に，上位機器とリーダライタの相互認証を実施します.
6	CaoExtension::Request

	カードの有無を確認します.
7	CaoExtension::Anticollision カードに衝突防止処理を行い, 確認できたカードのシリアル番号を取得します.
8	CaoExtension::Select 手順 7 で取得したシリアル No を引数に, カードを選択します.
9	CaoExtension::Authentication カードに対して相互認証処理をします.

3.3. コントローラクラス

3.3.1. ResetMain

リーダライタを初期化し，メインリセット以外のコマンドの実行を許可します．

メインリセットが実行されていない状態で，メインリセット以外のコマンドを実行するとエラーとなります．

メインリセットの実行が完了すると，上位機器との通信速度を初期値の 9600bps に設定します．

書式　　ResetMain()

戻り値　　：　[out] なし

使用例

```
‘ ResetMain
ctrl.Execute "ResetMain"
```

3.3.2. CheckCtrlState

リーダライタ内部の各制御部（リーダライタ制御部，カード通信制御部）が正常に起動しているかを確認します．正常に起動していることが確認された制御部に対しては，その制御部へのコマンドの実行を許可します．リーダライタ制御部のチェックを指定した場合は，リーダライタ制御部が正常に起動しているかの確認のみを行います．リーダライタ制御部のリセットコマンドは実行しません．このため，リーダライタ制御部の初期化も行いません．

カード通信制御部のチェックを指定すると，カード通信制御部のリセットコマンドを実行します．カード通信制御部は初期化され，リーダライタとカード間の通信設定は以下の初期値に戻ります．

項目	初期値
キャリア制御	キャリア出力停止
カードタイプ	TypeB
通信速度(リーダライター→カード)	106kbps
通信速度（カード→リーダライタ）	106kbps
カードからのレスポンス待ち時間	200ms

書式　　CheckCtrlState(<RWCtrl>, <CardCommCtrl>)

RWCtrl　　：　[in] リーダライタ制御部のチェックフラグ(VT_BOOL)

CardCommCtrl　　：　[in] カード通信制御部のチェックフラグ(VT_BOOL)

戻り値　　：　[out] なし

使用例

```
‘ ResetMain
ctrl.Execute "CheckCtrlState",Array(TRUE, TRUE)
```

3.3.3. GetReaderWriterInfo

リーダライタの固有情報を取得します．

書式　GetReaderWriterInfo ()

なし

戻り値　：　[out] メーカー情報バイト数(VT_UI1)
[out] ソフトウェアバージョン(4 バイト)(VT_ARRAY | VT_UI1)
[out] 製造情報(8 バイト)(VT_ARRAY | VT_UI1)

使用例

```
' ResetMain
ctrl.Execute "CheckCtrlState",Array(TRUE, TRUE)
```

GetReaderWriterInfo

構文　*object*.GetReaderWriterInfo

引数　なし

戻り値　<InfoCnt> = VT_UI1: メーカー情報バイト数
<SoftwareVer> = VT_ARRAY | VT_UI1: ソフトウェアバージョン(4 バイト)
<ProductInfo> = VT_ARRAY | VT_UI1: 製造情報(8 バイト)

説明　リーダライタの固有情報を取得します.

ModifyHostBaudRate

構文　*object*.ModifyHostBaudRate(<BaudRate>)

引数　<BaudRate> = VT_UI2: リーダライタから上位機器への通信速度

0	9600 kbps
1	19200 kbps
2	38400 kbps
3	614400 kbps
4	28800 kbps
5	57600 kbps
6	115200 kbps
7	230400 kbps
8	460800 kbps

戻り値 なし

説明 リーダライタの上位機器との通信速度を変更します.
このコマンドが成功した場合, リーダライタの通信速度が変更されるため, これまでの CaoController オブジェクトでは通信ができなくなります. 新しい通信速度を設定した CaoController オブジェクトを再生成してください.

SetLED

構文 *object*.SetLED(<State>, <OnInterval>, <OffInterval>)

引数 <State> = VT_UI2: LED の制御状態
0 デフォルト状態
1 LED 常時消灯
2 LED 常時点灯
3 LED 常時点滅
<OnInterval> = VT_UI2: LED 点滅時の消灯時間 (単位:ms)
<OffInterval> = VT_UI2: LED 点滅時の消灯時間 (単位:ms)

戻り値 なし

説明 リーダライタの LED 制御を行います.
点滅制御時は, 消灯 / 点灯時間を指定出来ます.
本コマンドにより設定された LED 制御状態は, パワーオンリセットもしくはメインリセットコマンドの実効で, デフォルト状態に戻ります.

3.4. 拡張ボードクラス

GetCardCommCtrlInfo

構文 *object*.GetCardCommCtrlInfo

引数 なし

戻り値 <InfoCnt> = VT_UI1: メーカー情報バイト数
<SoftwareVer> = VT_ARRAY | VT_UI1: ソフトウェアバージョン(4 バイト)
<ProductInfo> = VT_ARRAY | VT_UI1: 製造情報(8 バイト)

説明 リーダライタのカード通信制御部の情報を取得します.

SetCarrierState

構文 *object*.SetCarrierState(<State>)

引数 <State> = VT_UI1: キャリア出力

0 キャリア出力 OFF

1 キャリア出力 ON (通常出力)

2 キャリア出力 ON (高出力)

戻り値 なし

説明 リーダライタからキャリア出力の ON / OFF 状態を制御します.

キャリア出力 ON のとき, カードへキャリアを供給し, カードとの通信を可能にします.

キャリア出力 OFF のとき, キャリアを停止し, カードとの通信を終了します.

Authentication1

構文 *object*.Authentication1

引数 なし

戻り値 <Rnd1> = VT_ARRAY | VT_UI1: 認証用乱数1(8 バイト)

説明 リーダライタに対して認証用乱数1を要求します.

Authentication2

構文 *object*.Authentication2(<Rnd1>, <Rnd2>)

引数 <Rnd1> = VT_ARRAY | VT_UI1: 認証用乱数1(8 バイト)

Authentication1 コマンドで取得した認証用乱数を指定します.

<Rnd2> = VT_ARRAY | VT_UI1: 認証用乱数2(8 バイト)

戻り値 <Rnd2> = VT_ARRAY | VT_UI1: 認証用乱数2(8 バイト)

説明 上位機器とリーダライタの相互認証を実施します.

Authentication1 コマンドが実行済みのとき, このコマンドは使用可能になります.

Request

構文 *object*.Request(<Mode>)

引数 <Mode> = VT_UI1: リクエストコード

1 Request Std (Request)

2 Request All (WakeUp)

戻り値 <ATQ> = VT_ARRAY | VT_UI1: ATQ(2 バイト)

説明 カードに Request / WakeUp 処理を行って, カードの有無を確認します.

Anticollision

構文 *object*.Anticollision(<Level>, <BitCount>, <Snr>)

引数 <Level> = VT_UI1: カスケードレベル

1 カスケードレベル1

2 カスケードレベル2

<BitCount> = VT_UI1: カードのシリアル No.(Snr)のビット数

<Snr> = VT_ARRAY | VT_UI1: カードのシリアル No.(4 バイト)

BitCount で指定したビット数だけ, 既知のシリアル No.を設定します.

既知 Snr	Byte0			Byte1			Byte2			Byte3		
	B0	…	B7	B0	…	B7	B0	…	B7	B0	…	B7
BitCount	01h	…	08h	09h	…	10h	11h	…	18h	19h	…	20h

戻り値 <Snr> = VT_ARRAY | VT_UI1: カードのシリアル No.(4 バイト)

説明 カードに対し, 衝突防止処理を行い, カードのシリアル No.を取得します.
リーダライタとの通信可能範囲内に複数のカードが存在した場合は, 衝突防止処理により最初に確認したカードのシリアル No.のみを上位機器に送信します.

Select

構文 *object*.Select(<Level>, <Snr>)

引数 <Level> = VT_UI1: カスケードレベル

1 カスケードレベル1
2 カスケードレベル2

<Snr> = VT_ARRAY | VT_UI1: カードのシリアル No.(4 バイト)

戻り値 <SAK> = VT_UI1: SAK

説明 指定したシリアル No.を持つカードを選択します.
カードから受信した SAK を上位機器に送信します.
処理が正常に終了すると, カードは活性化されます.

Authentication

構文 *object*.Authentication(<Sector>, <Mode>, <Key>)

引数 <Sector> = VT_UI1: 認証対象の SectorNo.
Mifare Standard 1K カード 00h ～ 0Fh
Mifare Standard 4K カード 00h ～ 27h

<Mode> = VT_UI1: カスケードレベル
1 A キーで認証
2 B キーで認証

<Key> = VT_ARRAY | VT_UI1: Mifare 用認証キーデータ（6 バイト）

戻り値 なし

説明 Sector で指定したセクターに対して, Mode で指定した条件に従って, 相互認証処理を行います. 認証結果は, 上位機器に送信されます.

AuthenticationInMemory

構文 *object*.AuthenticationInMemory(<Sector>, <Mode>, <Key>)

引数 <Sector> = VT_UI1: 認証対象の SectorNo.
Mifare Standard 1K カード 00h ～ 0Fh
Mifare Standard 4K カード 00h ～ 27h

<Mode> = VT_UI1: 認証キーの A / B モード
1 A キーで認証
2 B キーで認証

<Key> = VT_UI1: Mifare 用不揮発性メモリのセクター番号（00h ～ 0Fh）

戻り値 なし

説明 Sector で指定したセクターに対して，Mode で指定した条件に従って，相互認証処理を行います．認証結果は，上位機器に送信されます．
認証キーデータの Mifare 用不揮発性メモリへの登録は，LoadKey コマンドを使用します．

Read

構文 *object*.Read(<Block>)

引数 <Block> = VT_UI1: 読み出しを行うカードのブロックアドレス
Mifare Standard 1K カード 00h ～ 3Fh
Mifare Standard 4K カード 00h ～ FFh

戻り値 <Data> = VT_ARRAY | VT_UI1: カードから読み出したデータ（16 バイト）

説明 Block で指定したブロックアドレスに格納されている 16 バイトのデータを読み出します．

Write

構文 *object*.Write(<Block>, <Data>)

引数 <Block> = VT_UI1: 読み出しを行うカードのブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FFh
<Data> = VT_ARRAY | VT_UI1: 書き込みデータ（16 バイト）

戻り値 なし

説明 Block で指定したブロックアドレスに，Data で指定した 16 バイトのデータを書き込みます．

DecrementTrans

構文 *object*.DecrementTrans(<DecBlock>, <TransBlock>, <Value>)

引数 <DecBlock> = VT_UI1: 減算対象となるデータが格納されているブロックアドレス

Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FFh

<TransBlock> = VT_UI1: 減算結果を格納するブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FEh
FFh を指定した場合は転送処理を行いません.

<TransBlock> = VT_UI4: 減算する数値

戻り値 なし

説明 指定したブロックアドレスの Value 形式の値に対し, 指定した値分の減算処理を実行し, 結果を指定したブロックアドレスに転送します.
減算処理するブロックアドレスに Value 形式のデータがない場合は, エラーを返します.
Value 形式のデータを設定する場合は, ValueWrite コマンドを使用してください.

IncrementTrans

構文 *object*.IncrementTrans(<IncBlock>, <TransBlock>, <Value>)

引数 <IncBlock> = VT_UI1: 加算対象となるデータが格納されているブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FFh

<TransBlock> = VT_UI1: 加算結果を格納するブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FEh
FFh を指定した場合は転送処理を行いません.

<TransBlock> = VT_UI4: 加算する数値

戻り値 なし

説明 指定したブロックアドレスの Value 形式の値に対し, 指定した値分の加算処理を実行し, 結果を指定したブロックアドレスに転送します.
加算処理するブロックアドレスに Value 形式のデータがない場合は, エラーを返します.
Value 形式のデータを設定する場合は, ValueWrite コマンドを使用してください.

RestoreTrans

構文 *object*.RestoreTrans(<ResBlock>, <TransBlock>)

引数

<ResBlock> = VT_UI1: リストア対象となるデータが格納されているブロックアドレス

Mifare Standard 1K カード 01h ～ 3Fh

Mifare Standard 4K カード 01h ～ FFh

<TransBlock> = VT_UI1: リストア結果を格納するブロックアドレス

Mifare Standard 1K カード 01h ～ 3Fh

Mifare Standard 4K カード 01h ～ FEh

FFh を指定した場合は転送処理を行いません.

戻り値

なし

説明

指定したブロックアドレスの Value 形式のデータを, 転送先に指定したブロックアドレスに上書きします.

リストア処理するブロックアドレスに Value 形式のデータがない場合は, エラーを返します.

Value 形式のデータを設定する場合は, ValueWrite コマンドを使用してください.

Transfer

構文

object.Transfer(<TransBlock>)

引数

<TransBlock> = VT_UI1: 転送対象データを格納するブロックアドレス

Mifare Standard 1K カード 01h ～ 3Fh

Mifare Standard 4K カード 01h ～ FFh

戻り値

なし

説明

DecrementTrans, IncremantTrans, RestoreTrans の各コマンドにおいて, 転送処理を実施しない条件(TransBlock に FFh を指定)でコマンド処理を行った場合に, カード内に保持している演算結果データを, 指定したブロックアドレスに転送します.

Halt

構文

object.Halt

引数

なし

戻り値

なし

説明

リーダライタの通信可能範囲内にある活性化されたカードを，ホルト状態にします．処理が正常に終了すると，カードは非活性化（コマンドに応答しない状態）されます．

AutoSelect

構文 *object*.AutoSelect(<Mode>)

引数 <Mode> = VT_UI1: リクエストコード

1 Request Std (Request)

2 Request All (WakeUp)

戻り値 <ATQ> = VT_ARRAY | VT_UI1: ATQ(2 バイト)

<Snr> = VT_ARRAY | VT_UI1: カードのシリアル No.(8 バイト)

<SAK> = VT_UI1: SAK

説明 カードに対するリクエスト処理から，セレクト処理までを一括して行います.

カードに対して，Request→Anticollision→Select を順次実行し，カードをアクティブ状態(活性化)にします.

AutoLogin

構文 *object*.AutoLogin(<ReqMode>, <Sector>, <AuthMode>, <Key>)

引数 <ReqMode> = VT_UI1: リクエストコード

1 Request Std (Request)

2 Request All (WakeUp)

<Sector> = VT_UI1: 認証対象の SectorNo.

Mifare Standard 1K カード 00h ～ 0Fh

Mifare Standard 4K カード 00h ～ 27h

<AuthMode> = VT_UI1: 認証キーの A / B モード

1 A キーで認証

2 B キーで認証

<Key> = VT_ARRAY | VT_UI1: Mifare 用認証キーデータ (6 バイト)

戻り値 <ATQ> = VT_ARRAY | VT_UI1: ATQ(2 バイト)

<Snr> = VT_ARRAY | VT_UI1: カードのシリアル No.(4 バイト)

<SAK> = VT_UI1: SAK

説明 カードに対するリクエスト処理から，相互認証処理までを一括して行います.

カードに対して，Request→Anticollision→Select→Authentication を順次実行し，カードをアクティブ状態(活性化)にします.

AutoLoginInMemory

構文 *object*.AutoLoginInMemory(<ReqMode>, <Sector>, <AuthMode>, <Key>)

引数 <ReqMode> = VT_UI1: リクエストコード

1 Request Std (Request)

2 Request All (WakeUp)

<Sector> = VT_UI1: 認証対象の SectorNo.

Mifare Standard 1K カード 00h ～ 0Fh

Mifare Standard 4K カード 00h ～ 27h

<AuthMode> = VT_UI1: 認証キーの A / B モード

1 A キーで認証

2 B キーで認証

<Key> = VT_UI1: Mifare 用不揮発性メモリのセクター番号 (00h ～ 0Fh)

戻り値 <ATQ> = VT_ARRAY | VT_UI1: ATQ(2 バイト)

<Snr> = VT_ARRAY | VT_UI1: カードのシリアル No.(4 バイト)

<SAK> = VT_UI1: SAK

説明 カードに対するリクエスト処理から, 相互認証処理までを一括して行います.
カードに対して, Request→Anticollision→Select→Authentication を順次実行し, カードをアクティブ状態(活性化)にします.
認証キーデータの Mifare 用不揮発性メモリへの登録は, LoadKey コマンドを使用します.

DecrementRead

構文 *object*.DecrementRead(<DecBlock>, <TransBlock>, <Value>)

引数 <DecBlock> = VT_UI1: 減算対象となるデータが格納されているブロックアドレス

Mifare Standard 1K カード 01h ～ 3Fh

Mifare Standard 4K カード 01h ～ FFh

<TransBlock> = VT_UI1: 減算結果を格納するブロックアドレス

Mifare Standard 1K カード 01h ～ 3Fh

Mifare Standard 4K カード 01h ～ FEh

FFh を指定した場合は転送処理を行いません.

<TransBlock> = VT_UI4: 減算する数値

戻り値 <Data> = VT_UI4: 減算結果の Value 値

説明 指定したブロックアドレスの Value 形式のデータに対し，減算処理を実行し，結果を指定したブロックのアドレスへ転送します．
転送処理後，転送先のアドレスのデータを読み出し，Value 値を返します．

IncrementRead

構文 *object*.IncrementRead(<IncBlock>, <TransBlock>, <Value>)

引数 <IncBlock> = VT_UI1: 加算対象となるデータが格納されているブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FFh
<TransBlock> = VT_UI1: 加算結果を格納するブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FEh
FFh を指定した場合は転送処理を行いません．
<TransBlock> = VT_UI4: 加算する数値

戻り値 <Data> = VT_UI4: 加算結果の Value 値

説明 指定したブロックアドレスの Value 形式のデータに対し，加算処理を実行し，結果を指定したブロックのアドレスへ転送します．
転送処理後，転送先のアドレスのデータを読み出し，Value 値を返します．

RestoreRead

構文 *object*.RestoreRead(<ResBlock>, <TransBlock>)

引数 <ResBlock> = VT_UI1: リストア対象となるデータが格納されているブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FFh
<TransBlock> = VT_UI1: リストア結果を格納するブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FEh
FFh を指定した場合は転送処理を行いません．

戻り値 <Data> = VT_UI4: 転送結果の Value 値

説明 指定したブロックアドレスの Value 形式データを，転送先に指定したブロックアドレスに上書きします．上書きされた転送先のデータを読み出し，Value 値を返します．

WriteRead

構文 *object*.WriteRead(<Block>, <Data>)

引数 <Block> = VT_UI1: 読み出しを行うカードのブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FFh
<Data> = VT_ARRAY | VT_UI1: 書き込みデータ（16 バイト）

戻り値 <Data> = VT_ARRAY | VT_UI1: 読み出しデータ（16 バイト）

説明 指定したブロックアドレスのデータに対し，指定したデータを書き込みます．
書き込み実行後，書き込みを行ったブロックアドレスのデータを読み出します．

ValueWrite

構文 *object*.ValueWrite(<Block>, <Data>)

引数 <Block> = VT_UI1: 読み出しを行うカードのブロックアドレス
Mifare Standard 1K カード 01h ～ 3Fh
Mifare Standard 4K カード 01h ～ FFh
<Data> = VT_UI4: Value 値

戻り値 <Data> = VT_UI4: Value 値

説明 指定した Value 値を 16 バイトの Value データ形式に変換して，指定のブロックアドレスに書き込みます．
書き込みを実行後，書き込みを行ったブロックアドレスのデータを読み出し，Value 値を返します．

LoadKey

構文 *object*.LoadKey(<Mode>, <Sector>, <Key>)

引数 <AuthMode> = VT_UI1: 認証キーの A / B モード

1 A キーで認証

2 B キーで認証

<Sector> = VT_UI1: Mifare 用不揮発性メモリのセクター番号(00h ～ 0Fh)

<Key> = VT_ARRAY | VT_UI1: Mifare 用認証キーデータ (6 バイト)

戻り値 なし

説明 Mifare 用認証キーをリーダライタ内の Mifare 用不揮発性メモリに登録します.

AutoRead

構文 *object*.AutoRead

引数 なし

戻り値 <Data> = VT_ARRAY | VT_UI1: カードから読み出したデータ (0～16 バイト)

説明 カードに対するリクエスト処理から, Read 処理までを一括して行います.
カードに対して, Request→Anticollision→Select→Authentication→Read を順次実行し,
カードの 1 ブロック分の情報を読み出します.