WB－1J

アプリケーションインターフェース
マニュアル
ＲＦＩＤ編

(Version 1.08)

改訂履歴

版	日付	変更内容	担当
1.00	2010/12/20	初版作成	
1.01	2011/1/20	RFID_ScanPICC 関数の WaitTime パラメータの単位が誤っていたのを修正。	
1.02	2011/1/27	誤字修正	
1.03	2011/4/7	2.2　アプリケーションタイプに MIFARE Ultralight C、MIFARE Plus を追記 2.3　RFID_SamAuthentication の追記 2.4　RFID-API コールシーケンス例のパターン追記 2.6　１）　RFID_Open 失敗時の返り値が誤っていたのを修正 2.6　６）、７）　アプリケーションに MIFARE Plus を追記 2.6　８）　FeliCa 相当以外の場合、lpResponseBuffer に NULL を設定する記載が誤っていただめ修正 2.6　１）～１０）　RFID-API コールシーケンス例にあわせて、パターンによって使用する必要のない API は、その旨を追記 2.6　７）～１０）　lpstatus に 0xC0xx を検出したときの対処方法の記載が誤っていたため修正。また、0xC0xx を検出する際の発生条件を補足 2.7.1　認証パラメータで、暗号化モードに関する記載不足を追記 2.10　１）API ステータスコードにて、0x2000 の記載漏れ追加、0x2000～0x2002 の意味が誤っていただめ修正 2.10　４）ISO25693 対応モジュールの上位ステータスコードの 0x40 は 0x00 の誤りであったため修正 その他誤記修正と補足説明の追加	
1.04	2011/4/22	2.6　１）～１１）　を 2.6.1～2.6.11 へ表記を変更。及び目次への反映 2.4　シーケンス例において、AutoSelectTarget を使用すべき場所の記載が無かったため追加（複数 PICC 時のアプリの指定に追加） 2.4.6、2.4.7 で、WB-1JR、WB-1JN 共用で動作するアプリケーションを作成する際には、TAG を使用時にも本 API を使用できるため、「使用する必要はありません。」の記述を変更 2.6.8　FeliCa 相当以外の場合、nResponseBufferSize のチェックは行っていないため、0 も指定できることを追記 2.6.11　本 API 以外で SAM 通信回数がクリアされる場合を記載 その他補足説明の追加	
1.05	2011/6/9	2.6.6　lpstatus に 0xC0xx を検出することがあるのを追記 2.6.11　注１の記載内容に、RFID_AutoSelectTarget が不足していたのを追記	
1.06	2011/10/11	2.6.4　RFID_ScanPICC の注意事項追記 2.6.5　RFID_ScanPICC2 の追加 2.6.10　RFID_Write の注意点追記 RFID_ScanPICC2 追加に伴う関連項目への追記 その他誤記修正	

版	日付	変更内容	担当
1.07	2011/12/22	2.4.3、2.4.4　単一 PICC の場合にアプリの指定で AutoSelectTarget の記載が漏れていたのを追記 2.4.1～2.4.4 において、リード/ライトをループする際に、それぞれの場合に応じた戻り先を追記	
1.08	2012/11/26	2.6.2　RFID_Close に注意事項追記	

― 目次 ―

1　はじめに

1.1　目的

本書は、WB-1J に内蔵された ISO/IEC14443 に対応した NFC カードリーダーライターモジュール、又は、ISO/IEC15693 に対応した TAG カードリーダーライターモジュールを利用するアプリケーションを作成する際に使用する、ＲＦＩＤアプリケーションインターフェースについて記述したものである。

1.2　適用範囲

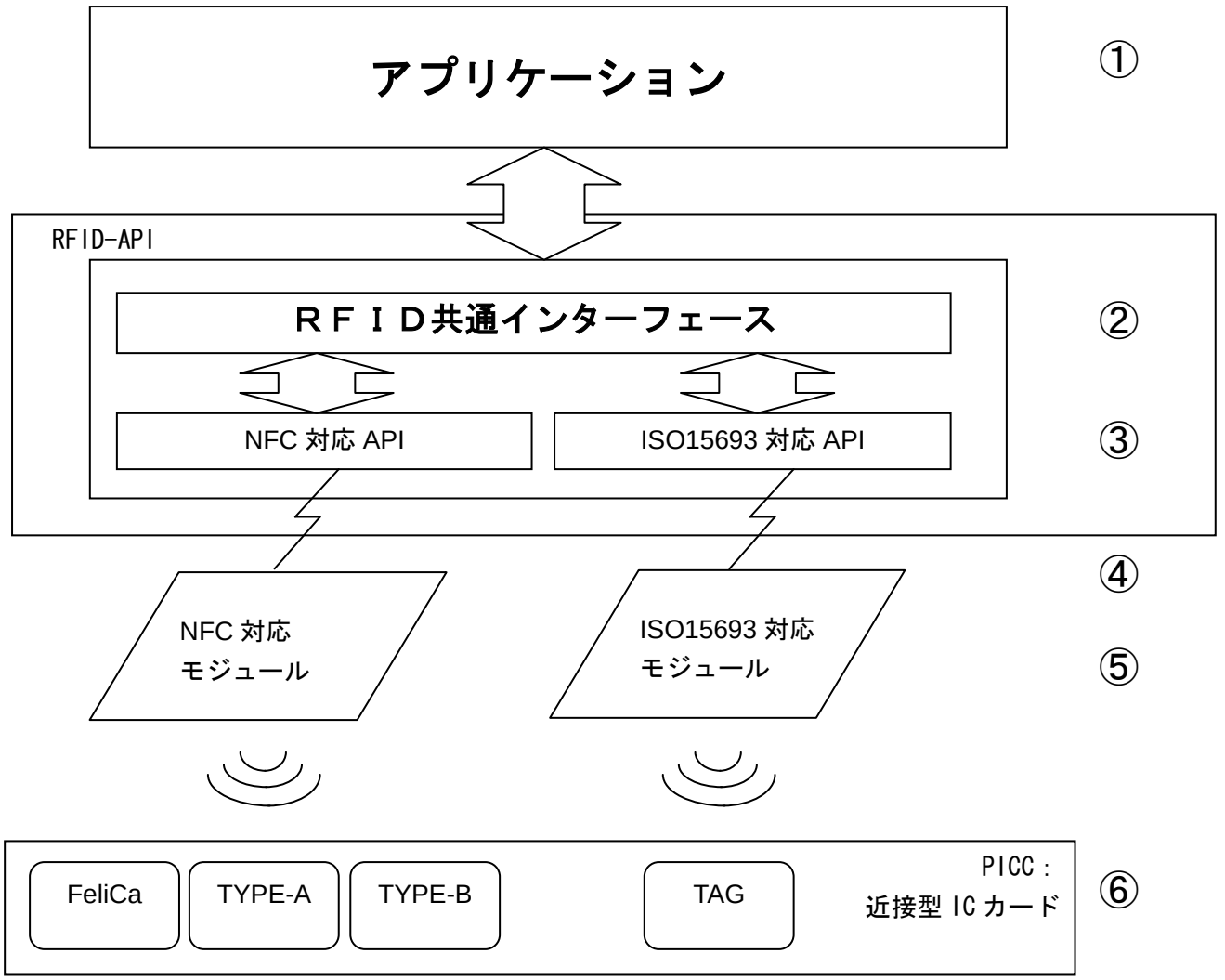
WB-1J に搭載されるＲＦＩＤアプリケーションインターフェースに適用する。

2　ＲＦＩＤアプリケーションインターフェース仕様

2.1　ＲＦＩＤアプリケーションインターフェースの機能

当ＡＰＩをアプリケーションプログラムに組み込む事により開発者は、リーダーライターモジュールによる WB-1J～リーダーライターモジュール間の通信プロトコル、及び、コマンドの違いを意識する事なく、リーダーライターモジュールに対してカード操作を行う事が出来る。

カード操作については、カードの各仕様及び、リーダーライターモジュールの各コマンド仕様を理解している必要がある。



① 開発するカードアプリケーション
② アプリケーションからコールされる RFID の共通インターフェース
③ 各リーダーライターモジュールのコマンドに１対１で対応したインターフェース
※NFC と TAG を同時に使用することは出来ません。
④ シリアル回線
⑤ NFC 対応モジュール、ISO15693 対応モジュール
⑥ FeliCa、TYPE-A、TYPE-B、TAG の各カードタイプ

2.2　ＰＩＣＣとカードアプリケーションの概要

PICC（近接型 IC カード）とカード内部アプリケーションの関係を以下の表に示す。
カードデータにアクセスするには、PICC の選択とアプリケーションの選択が必要である。

PICC タイプ	アプリケーションタイプ
TypeA	MIFARE Classic
	MIFARE Ultralight
	MIFARE Ultralight C
	MIFARE DESFire(ISO 準拠)
	MIFARE Plus
TypeB	標準公開されているアプリケーションは無い （カードの仕様に従い、低レベル API での開発が必要。低レベル API の開示を要望される場合は、別途お問合せください。）
FeliCa	FeliCa
	SSFC

TAG

PICC タイプ	アプリケーションタイプ
ISO15693（TAG）	ISO15693（Tag-it HF-I Plus/Pro、I-CODE SLI）

※ISO15693（TAG）は、Tag-it HF-I Plus/Pro、I-CODE SLI の区別はリーダ側ではできません。

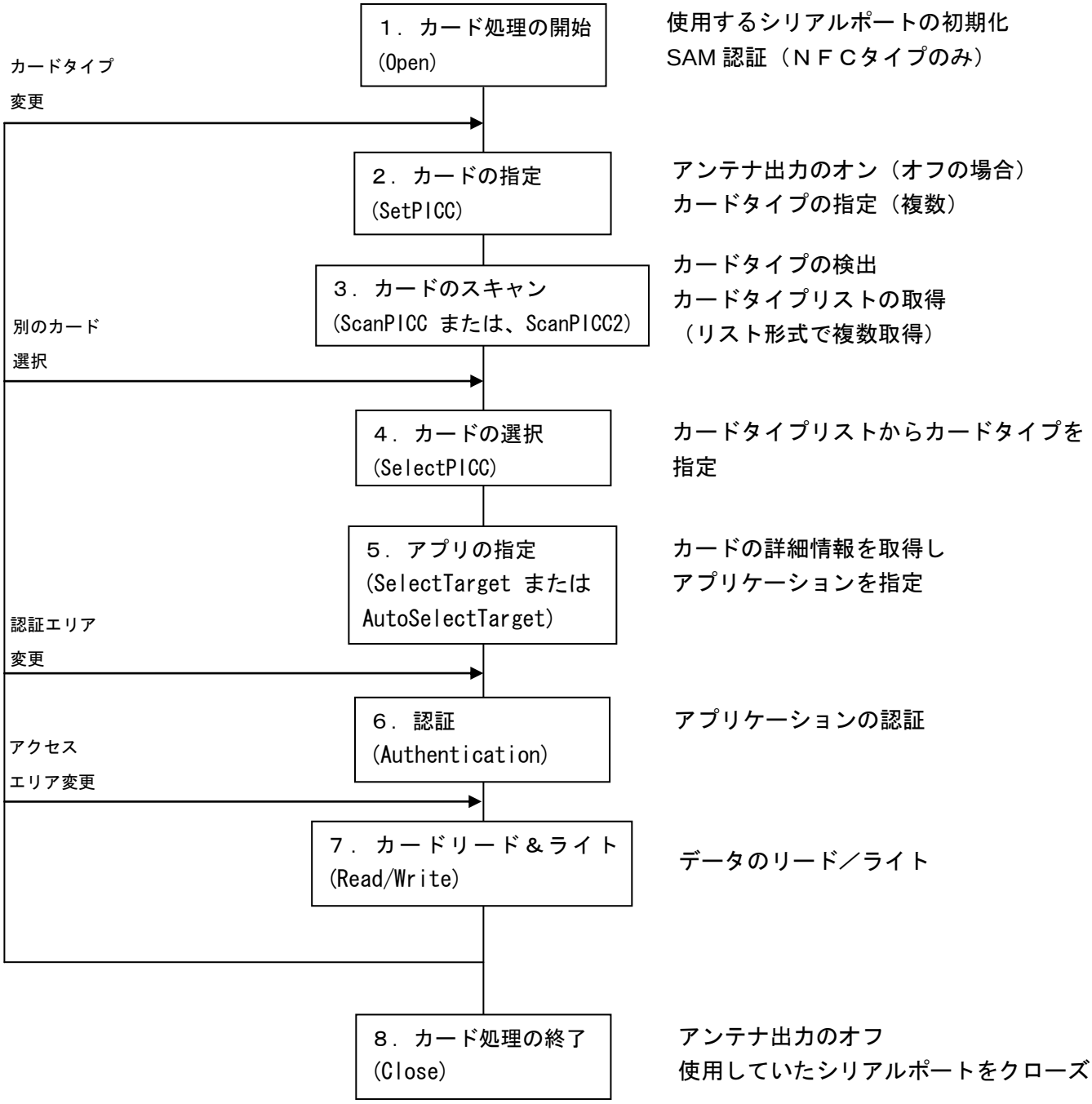
2.3　ＲＦＩＤアプリケーションインターフェース

RFID API として、標準ストリーム入出力 API を用意する。
ユーザは下記 API を組み合わせ、カードアプリケーションを作成する。

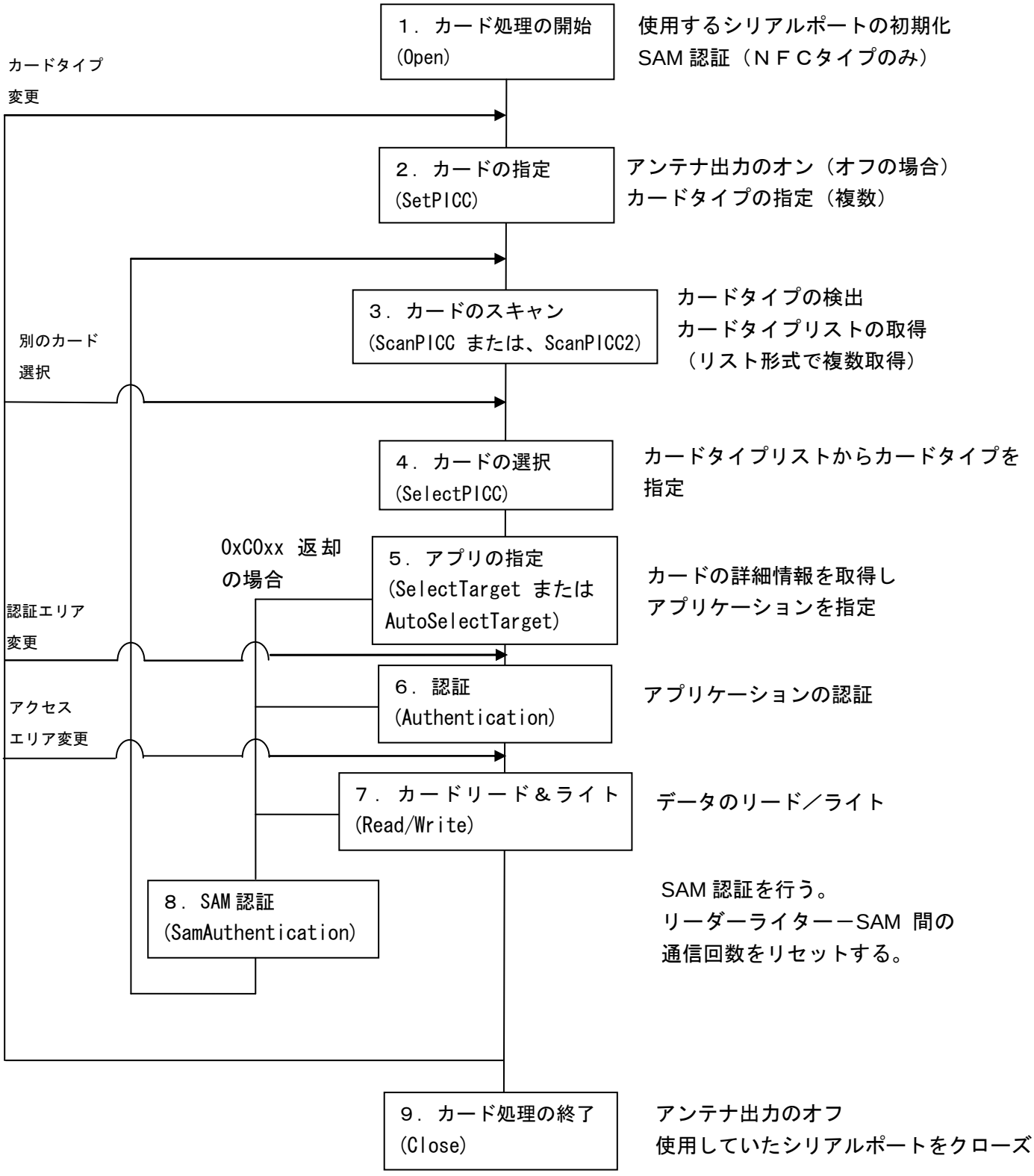
No.	API 名称	機能
1	RFID_Open	リーダーライターモジュールへのアクセスを開始する パラメータにより、リーダーライターの種別を選択する
2	RFID_Close	リーダーライターモジュールへのアクセスを停止する
3	RFID_SetPICC	対象とする PICC を選択する アンテナ出力を開始する
4	RFID_ScanPICC	PICC をスキャンする
5	RFID_ScanPICC2	PICC をスキャンする ※RFID_ScanPICC より精度の高い待ち時間制御を行う
6	RFID_SelectPICC	PICC を選択する
7	RFID_AutoSelectTarget	アプリケーションの自動選択を行い、アプリケーション毎の情報を取得する ※複数のアプリケーションを使用する場合に使用する為、 アプリケーションを固定する運用では使用しない
8	RFID_SelectTarget	アプリケーションを特定する
9	RFID_Authentication	PICC 及びアプリケーションの認証を行う
10	RFID_Write	カードへデータを書き込む
11	RFID_Read	カードからデータを読み出す
12	RFID_SamAuthentication	SAM 認証を行う。リーダーライターーＳＡＭ間の通信回数をリセットする。

2.4　　シーケンス例

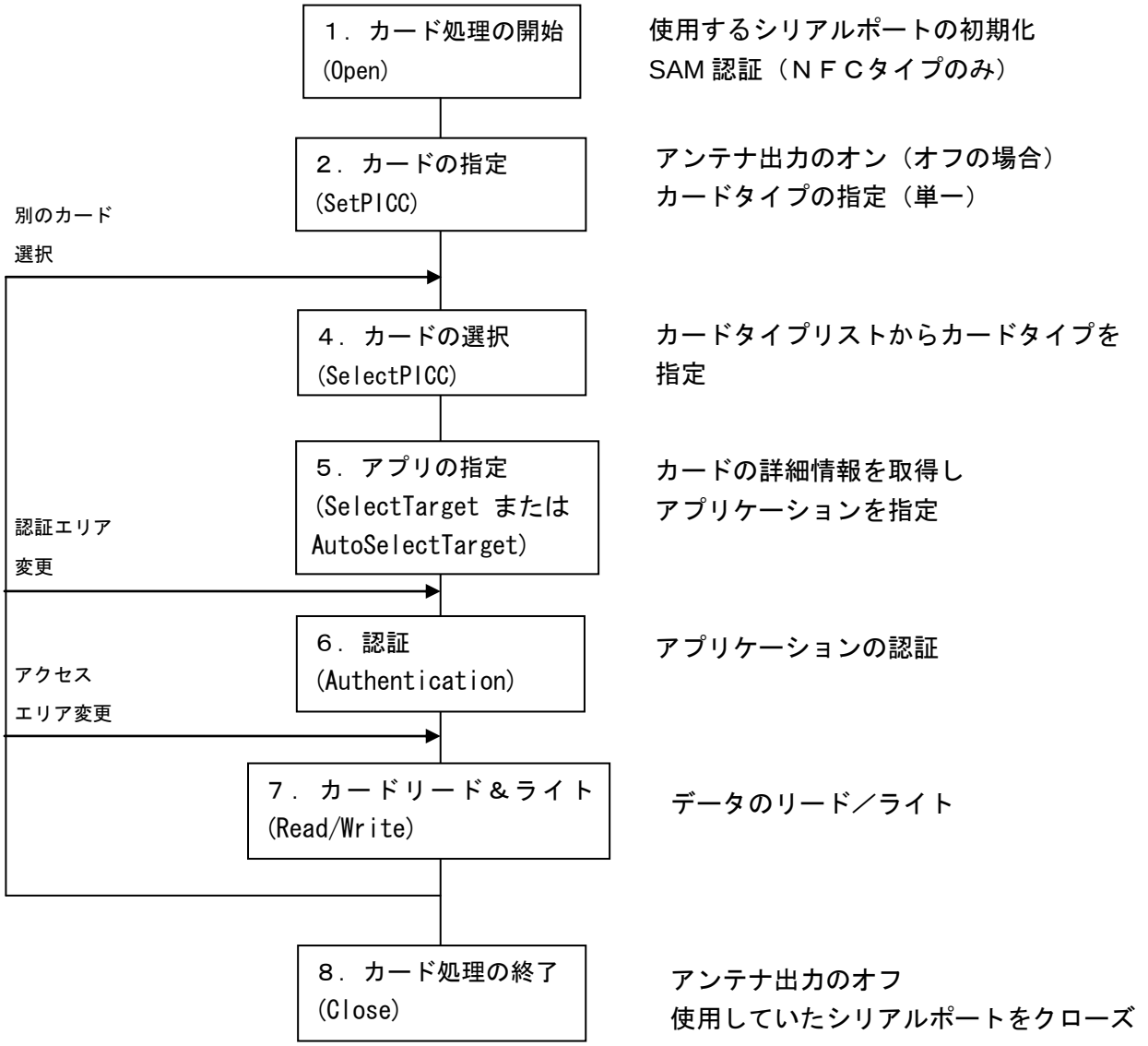
2.4.1　ＲＦＩＤ－ＡＰＩコールのシーケンス例（複数 PICC の場合）



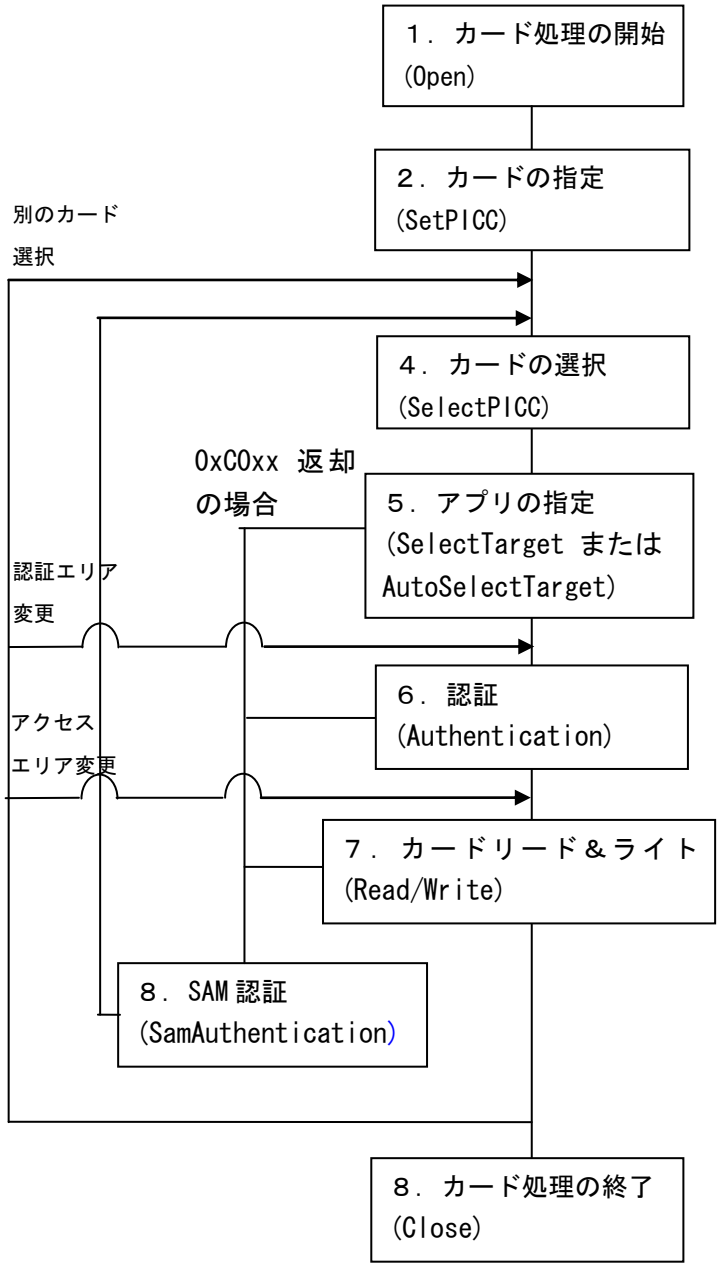
2.4.2　ＲＦＩＤ－ＡＰＩコールのシーケンス例（複数 PICC、FeliCa 相当の場合）



2.4.3　ＲＦＩＤ－ＡＰＩコールのシーケンス例（単一 PICC の場合）



2.4.4　ＲＦＩＤ－ＡＰＩコールのシーケンス例（単一 PICC、FeliCa 相当の場合）



使用するシリアルポートの初期化
SAM 認証（ＮＦＣタイプのみ）

アンテナ出力のオン（オフの場合）
カードタイプの指定（単一）

カードタイプリストからカードタイプを
指定

カードの詳細情報を取得し
アプリケーションを指定

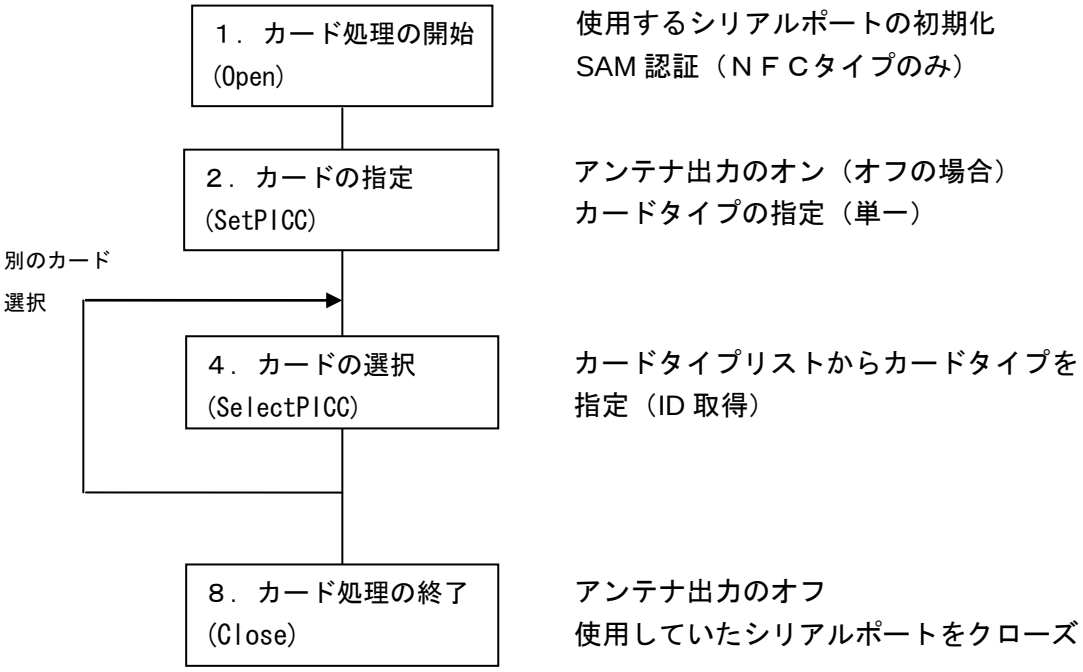
アプリケーションの認証

データのリード／ライト

SAM 認証を行う。
リーダーライター－SAM 間の
通信回数をリセットする。

アンテナ出力のオフ
使用していたシリアルポートをクローズ

2.4.5　ＲＦＩＤ－ＡＰＩコールのシーケンス例（単一 PICC、ID 運用の場合）



2.5　ＤＬＬ呼び出し方法

RFID API は、「rfidapi.dll」内に存在する。API をコールする際の、ＤＬＬ呼び出し方法例として、RFID 共通インターフェースをコールする際の関数ポインタ取得方法を下記に記す。
なお、.NET アプリケーションの場合は、dllimport を使用することで、下記は不要となる。

```
// 型定義
typedef RFID (__stdcall * RFID_Open)(int DevType,LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_Close)(RFID *hRFID,LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_SetPICC)(RFID *hRFID,BYTE Picc,LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_ScanPICC)(RFID *hRFID,LPBYTE lpPiccTypeBuffer,
                    DWORD nPiccTypeBufferSize, long WaitTime, LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_ScanPICC2)(RFID *hRFID,LPBYTE lpPiccTypeBuffer,
                    DWORD nPiccTypeBufferSize, long WaitTime, LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_SelectPICC)(RFID *hRFID,BYTE Id,WORD SystemCode,
                    LPBYTE lpIdBuffer, DWORD nIdBufferSize, LPDWORD lpnIdSize,
                    LPBOOL lpAntiCollision, LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_AutoSelectTarget)(RFID *hRFID,BYTE Id, LPBYTE lpKindBuffer,
                    DWORD nKindBufferSize, LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_SelectTarget)(RFID *hRFID,BYTE Id,BYTE Kind,LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_Authentication)(RFID *hRFID,
                    RFID_AUTHENTICATION_DATA* lpAuthParam,
                    LPBYTE lpAuthData, LPBYTE lpResponseBuffer, DWORD nResponseBufferSize,
                    LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_Write)(RFID *hRFID, RFID_RW_PARAM *lpRW_Param,
                    LPBYTE lpWriteBuffer, DWORD nWriteBufferSize, LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_Read)(RFID *hRFID, RFID_RW_PARAM *lpRW_Param,
                    LPBYTE lpReadBuffer, DWORD nReadBufferSize, LPDWORD lpStatus);
typedef BOOL (__stdcall * RFID_SamAuthentication)(RFID *hRFID, BYTE mode, LPBYTE datap,
                    DWORD data_len, LPDWORD lpStatus);

// 関数ポインタ変数
RFID_Open rfid_Open;
RFID_Close rfid_Close;
RFID_SetPICC rfid_SetPICC;
RFID_ScanPICC rfid_ScanPICC;
RFID_ScanPICC2 rfid_ScanPICC2;
RFID_SelectPICC rfid_SelectPICC;
RFID_AutoSelectTarget rfid_AutoSelectTarget;
RFID_SelectTarget rfid_SelectTarget;
RFID_Authentication rfid_Authentication;
RFID_Write rfid_Write;
RFID_Read rfid_Read;
```

```
RFID_SamAuthentication rfid_SamAuthentication;HMODULE hDll = LoadLibrary(L"rfidapi.dll");

if (hDll != NULL) {
    rfid_Open = (RFID_Open) GetProcAddress(hDll, L"RFID_Open");
    rfid_Close = (RFID_Close) GetProcAddress(hDll, L"RFID_Close");
    rfid_SetPICC = (RFID_SetPICC) GetProcAddress(hDll, L"RFID_SetPICC");
    rfid_ScanPICC = (RFID_ScanPICC) GetProcAddress(hDll, L"RFID_ScanPICC");
    rfid_ScanPICC2 = (RFID_ScanPICC) GetProcAddress(hDll, L"RFID_ScanPICC2");
    rfid_SelectPICC = (RFID_SelectPICC) GetProcAddress(hDll, L"RFID_SelectPICC");
    rfid_AutoSelectTarget = (RFID_AutoSelectTarget) GetProcAddress(hDll,
                                        L"RFID_AutoSelectTarget");
    rfid_SelectTarget = (RFID_SelectTarget) GetProcAddress(hDll, L"RFID_SelectTarget");
    rfid_Authentication = (RFID_Authentication) GetProcAddress(hDll, L"RFID_Authentication");
    rfid_Write = (RFID_Write) GetProcAddress(hDll, L"RFID_Write");
    rfid_Read = (RFID_Read) GetProcAddress(hDll, L"RFID_Read");
    rfid_SamAuthentication = (RFID_SamAuthentication) GetProcAddress(hDll,
                                        L"RFID_SamAuthentication");

}
```

2.6　ＲＦＩＤ共通インターフェース

2.6.1　オープン

リーダーライターモジュールへのアクセスを開始する。

呼び出し形式：
RFID * RFID_Open (int DevType, LPDWORD lpStatus);

引数：

DevType	1: NFC モジュール,2: ISO15693 モジュール
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：
成功した場合は、リーダーライターのハンドルを返す。
失敗した場合は、NULL を返す。

2.6.2　クローズ

リーダーライターモジュールへのアクセスを終了する。

呼び出し形式：
BOOL RFID_Close (RFID *hRFID, LPDWORD lpStatus);

引数：

hRFID	オープン時に取得したハンドル
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：
FALSE：クローズ失敗、TRUE:クローズ成功

注１） RFID_Close を呼び出すと、リーダーライターモジュールの電源が OFF になりますが、完全に電源が OFF になるまで 200ms を要します。再度 RFID_Open を呼び出す場合、200ms 以上時間を空けて呼び出す必要があります。RFID_Close の後、時間を空けずに RFID_Open を呼び出すと処理速度が低下することがあります。

2.6.3　ＰＩＣＣタイプの設定

対象となるＰＩＣＣタイプを選択し、通信準備を行う。

呼び出し形式：

```
BOOL RFID_SetPICC (RFID *hRFID, BYTE Picc, LPDWORD lpStatus);
```

引数：

hRFID	オープン時に取得したハンドル
Picc	ＰＩＣＣタイプ 0x00：指定なし、サポートする全ての PICC タイプが対象となる。 TAG の場合は 0x00 のみとする。 0x01：FeliCa 通信相当 0x02：Type A 0x03：Type B ※0x00 以外を指定した場合は、本 API コール後 RFID_SelectPICC をコールして下さい。RFID_ScanPICC を使用する必要はありません。
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：ＰＩＣＣタイプの設定失敗、TRUE: ＰＩＣＣタイプの設定成功

2.6.4　ＰＩＣＣタイプのスキャン

ＰＩＣＣタイプのスキャンを行う。

呼び出し形式：

```
BOOL RFID_ScanPICC (RFID *hRFID, LPBYTE lpPiccTypeBuffer, DWORD nPiccTypeBufferSize,
                                                    long WaitTime,LPDWORD lpStatus);
```

引数：

hRFID　　オープン時に取得したハンドル

lpPiccTypeBuffer　　ＰＩＣＣタイプリストを格納する為のバッファへのポインタ

オフセット	項目	意　味
0	リスト番号	0x01～
1	固定値	0x02
2	デバイス	CONFIGURATION のデバイス（未使用）
3	PICC タイプ	0x01:FeliCa 相当、0x02:TypeA 0x03:TypeB、0x00:TAG

PICC タイプ分繰り返し

nPiccTypeBufferSize　　ＰＩＣＣタイプリストを格納する為のバッファのサイズを指定する

WaitTime　　ＰＩＣＣタイプの検出をタイムアウトするまでの時間(単位:100 ミリ秒)、
※ 0 を指定するとＰＩＣＣを検出するまで、当ＡＰＩから復帰しない為、注意が必要です。
※ 1 を指定した場合でもカードの有無確認に最大 160ms 程度かかるためその間は当 API から復帰しないため、注意が必要です。
※ [有効範囲] 0～42949672

lpStatus　　ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：カード未検出、TRUE: カード検出

注２） FeliCａ系で、暗号化領域をリード／ライトした場合は、次に別のカードを認識するまで、カード未検出となります。

注３） RFID_SetPICC で PICC タイプに 0x00 以外を指定した場合は、本 API を使用する必要はありません。

注４） 本 API は、指定した WaitTime よりも実際にタイムアウトするまでの時間が長くなります。この誤差は、WaitTime の設定値を大きくするほど、誤差も大きくなります。精度の高いタイムアウトが必要な場合は、RFID_ScanPICC2 を利用してください。

2.6.5　ＰＩＣＣタイプのスキャン（高精度タイムアウト付き）

RFID_ScanPICC API より高精度なタイムアウトを用いてＰＩＣＣタイプのスキャンを行う。

呼び出し形式：

```
BOOL RFID_ScanPICC2 (RFID *hRFID, LPBYTE lpPiccTypeBuffer, DWORD nPiccTypeBufferSize,
                                                    long WaitTime, LPDWORD lpStatus);
```

引数：

hRFID　　オープン時に取得したハンドル

lpPiccTypeBuffer　　ＰＩＣＣタイプリストを格納する為のバッファへのポインタ

オフセット	項目	意　味
0	リスト番号	0x01～
1	固定値	0x02
2	デバイス	CONFIGURATION のデバイス（未使用）
3	PICC タイプ	0x01:FeliCa 相当、0x02:TypeA 0x03:TypeB、0x00:TAG

PICC タイプ分繰り返し

nPiccTypeBufferSize　　ＰＩＣＣタイプリストを格納する為のバッファのサイズを指定する

WaitTime　　ＰＩＣＣタイプの検出をタイムアウトするまでの時間(単位:100 ミリ秒)、
※ 0 を指定するとＰＩＣＣを検出するまで、当ＡＰＩから復帰しない為、注意が必要です。
※ 1 を指定した場合でもカードの有無確認に最大 160ms 程度かかるためその間は当 API から復帰しないため、注意が必要です。
※ ［有効範囲］0～42949672

lpStatus　　ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：カード未検出、TRUE: カード検出

注１） FeliCａ系で、暗号化領域をリード／ライトした場合は、次に別のカードを認識するまで、カード未検出となります。

注２） RFID_SetPICC で PICC タイプに 0x00 以外を指定した場合は、本 API を使用する必要はありません。

2.6.6　ＰＩＣＣタイプの選択

設定したＰＩＣＣタイプ、もしくはスキャンしたＰＩＣＣタイプを指定し、詳細情報を取得する。

呼び出し形式：

```
BOOL RFID_SelectPICC (RFID *hRFID, BYTE Id,WORD SystemCode, LPBYTE lpIdBuffer,
            DWORD nIdBufferSize, LPDWORD lpnIdSize, LPBOOL lpCollision,LPDWORD lpStatus);
```

引数：

hRFID	オープン時に取得したハンドル
Id	ＰＩＣＣリスト中のリスト番号を指定する。 ※事前に RFID_ScanPICC をコールした場合のみ、 RFID_SetPICC で単一の PICC を設定した場合は 0 を指定して下さい。
SystemCode	FeliCa システムコードを指定する ※指定が不要な場合やその他のカード使用時は 0x0000 を指定する。
lpIdBuffer	詳細情報を格納する為のバッファへのポインタ

詳細情報（TAG 以外の場合）

オフセット	サイズ	項目	意　味	
0	1	IDn	0x01～	検出分繰り返し
1	1	長さ	IDn の PICC 情報の長さ	
2		IDn の PICC 情報	Type A、Type B、FeliCa SRIX4K、TAG の各 PICC 別の情報（①～⑤）	

IDn の PICC 情報

①Type A

オフセット	サイズ	項目	意　味
2	2	ATQA	各 PICC の仕様書参照
4	1	SAK	各 PICC の仕様書参照
5	1	LEN	シングル：４、ダブル：７ トリプル：１０
6	4,7,10	UID	各 PICC の仕様書参照

②Type B

オフセット	サイズ	項目	意　味
2	4	PUPI	各 PICC の仕様書参照
6	4	アプリケーションデータ	各 PICC の仕様書参照
10	3	プロトコル情報	各 PICC の仕様書参照

③FeliCa

オフセット	サイズ	項目	意　味
2	8	IDm	各 PICC の仕様書参照
10	8	PMm	各 PICC の仕様書参照

④SRIX4K

オフセット	サイズ	項目	意　味
2	8	UID	各 PICC の仕様書参照

*SRIX4K は Type B ではあるが UID を持つ為、当 API では Type　B とは別扱とする

⑤TAG

オフセット	サイズ	項目	意　味	
0	1	DSFID	各 PICC の仕様書参照	検出分
1	8	UID	各 PICC の仕様書参照	繰り返し

nIdBufferSize	詳細情報を格納する為のバッファのサイズを指定する
lpnIdSize	取得した詳細情報のサイズを格納するためのバッファへのポインタ
lpCollision	FALSE:コリジョン未検出、TRUE:コリジョン検出（詳細情報が複数あり）
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：ＰＩＣＣタイプの選択失敗、TRUE: ＰＩＣＣタイプの選択成功

注）各ＰＩＣＣの仕様書に関しては、ＰＩＣＣ事業者にお問合せください。

2.6.7　アプリケーションの自動選択

指定したＰＩＣＣタイプからアプリケーションを自動選択し、カード種別を取得する。

呼び出し形式：

```
BOOL RFID_AutoSelectTarget (RFID *hRFID,BYTE Id,LPBYTE lpKindBuffer,
                                                DWORD nKindBufferSize, LPDWORD lpStatus);
```

引数：

hRFID	オープン時に取得したハンドル
Id	RFID_SelectPICC で取得した詳細情報の IDn を指定する。
lpKindBuffer	アプリケーション情報を格納する為のバッファへのポインタ

オフセット	サイズ	項目	意　味
0	1	IDn	0x01～
1	1	長さ	種別の長さ
2	1	アプリケーション （カード種別）	0x00:未認識または TAG 0x10:FeliCa 相当 0x16:SSFC 0x20:Type A 相当 0x21:MIFARE Ultralight 0x22:DESFire 0x23:MIFARE Classic 0x25:MIFARE Ultralight C 0x26:MIFARE Plus 0x30:Type B 相当 0x31:SRIX4K

カード種別分繰り返し

nKindBufferSize	アプリケーション（カード種別）を格納する為のバッファのサイズを指定する
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：アプリケーションの自動選択失敗、TRUE: アプリケーションの自動選択成功

注１）アプリケーションが FeliCa 相当の場合：ステータスコード 0xC0xx を検出した場合、RFID_SamAuthentication を行い､RFID_ScanPICC もしくは RFID_SelectPICC からやり直して下さい。
アプリケーションが FeliCa 相当以外の場合：ステータスコード 0xC0xx を検出することはありません。

注２）TAG のみを使用する場合は、アプリケーションが固定となる為、本 API を使用する必要はありません。

注３）カード内にアプリケーションが複数ある場合は最初に見つかったアプリケーションが選択されます。

2.6.8　アプリケーションの選択

アプリケーションを指定する。

呼び出し形式：

```
BOOL RFID_SelectTarget(RFID *hRFID, BYTE Id, BYTE Kind, LPDWORD lpStatus);
```

引数：

hRFID	オープン時に取得したハンドル
Id	RFID_SelectPICC で取得した詳細情報の IDn を指定する。
Kind	アプリケーション（カード種別）を指定する。

アプリケーション （カード種別）	0x00:TAG 0x10:FeliCa 相当 0x16:SSFC 0x20:Type A 相当 0x21:MIFARE Ultralight 0x22:DESFire 0x23:MIFARE Classic 0x25:MIFARE Ultralight C 0x26:MIFARE Plus 0x30:Type B 相当 0x31:SRIX4K

lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：アプリケーションの選択失敗、TRUE: アプリケーションの選択成功

注１）アプリケーションが FeliCa 相当の場合：ステータスコード 0xC0xx を検出した場合、RFID_SamAuthentication を行い、RFID_ScanPICC もしくは RFID_SelectPICC からやり直して下さい。
アプリケーションが FeliCa 相当以外の場合：ステータスコード 0xC0xx を検出することはありません。

注２）TAG のみを使用する場合は、アプリケーションが固定となる為、本 API を使用する必要はありません。

2.6.9　アプリケーションの認証

選択したアプリケーションに対して、認証処理を行う。

呼び出し形式：

```
BOOL RFID_Authentication (RFID *hRFID, RFID_AUTHENTICATION_DATA* lpAuthParam ,
            LPBYTE lpAuthData, LPBYTE lpResponseBuffer, DWORD nResponseBufferSize,
            LPDWORD lpStatus)
```

引数：

hRFID	オープン時に取得したハンドル
lpAuthParam	認証パラメータ（２．７．１を参照）
lpAuthData	SRIX4K、MIAFRE Ultralight は認証データが可変長である為、認証データを格納する為のバッファへのポインタを指定する。 ※認証パラメータは STANDARD を使用し、ここで確保したバッファサイズを指定する事。 他の種別のカードの場合は、NULL を指定する。
lpResponseBuffer	レスポンスデータ用バッファへのポインタを指定する。 アプリケーションが FeliCa 相当の場合、認証後、IDt が設定される。 FeliCa 相当以外の場合は何も設定されませんが、NULL は指定しないで下さい。
nResponseBufferSize	レスポンスデータを格納する為のバッファのサイズを指定する。 必ず１以上を指定して下さい。 ただし、FeliCa 相当以外の場合はサイズのチェックを行いませんので、その場合は、0 でも構いません。
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：アプリケーションの認証失敗、TRUE: アプリケーションの認証成功

注１）認証データは、PICC、アプリケーション毎に異なります。

注２）アプリケーションが FeliCa 相当の場合：ステータスコード 0xC0xx を検出し、
lpResponseBuffer にデータが格納されない場合、RFID_SamAuthentication を行い、
RFID_ScanPICC もしくは RFID_SelectPICC からやり直して下さい。
アプリケーションが FeliCa 相当以外の場合：ステータスコード 0xC0xx を検出することはありません。

注３）TAG のみを使用する場合は、アプリケーションが固定となる為、本 API を使用する必要はありません。

2.6.10　ＰＩＣＣへのライト

認証したアプリケーションに対して、書き込みを行う。

呼び出し形式：

```
BOOL RFID_Write (RFID *hRFID, RFID_RW_PARAM *lpRW_Param,LPBYTE lpWriteBuffer,
                                          DWORD nWriteBufferSize,LPDWORD lpStatus) ;
```

引数：

hRFID	オープン時に取得したハンドル
lpRW_Param	リード／ライトパラメータ（２．７．２を参照）
lpWriteBuffer	ライトデータを格納するバッファへのポインタ
nWriteBufferSize	ライトデータのサイズ
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：ライト失敗、TRUE: ライト成功

注１）ライトするデータは、PICC、アプリケーション毎に異なります。

注２）アプリケーションがFeliCa相当の場合：ステータスコード0xC0xxを検出した場合、
RFID_SamAuthenticationを行い、RFID_ScanPICCもしくはRFID_SelectPICCから
やり直して下さい。
アプリケーションがFeliCa相当以外の場合：ステータスコード0xC0xxを検出することはありません。

注３）ライトに失敗した場合、対象範囲内のデータの内容は保証されません。

2.6.11　ＰＩＣＣからのリード

認証したアプリケーションに対して、読み込みを行う。

呼び出し形式：

```
BOOL RFID_Read (RFID *hRFID, RFID_RW_PARAM *lpRW_Param, LPBYTE lpReadBuffer,
                                                    DWORD nReadBufferSize,LPDWORD lpStatus) ;
```

引数：

hRFID	オープン時に取得したハンドル
lpRW_Param	リード／ライトパラメータ（２．７．２を参照）
lpReadBuffer	リードデータを格納するバッファへのポインタ
nReadBufferSize	リードデータのサイズ
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：

FALSE：リード失敗、TRUE: リード成功

注１）リードするデータは、PICC、アプリケーション毎に異なります。

注２）アプリケーションがFeliCa相当の場合：ステータスコード0xC0xxを検出し、
lpReadBufferにデータが格納されない場合、RFID_SamAuthenticationを行い、
RFID_ScanPICCもしくはRFID_SelectPICCからやり直して下さい。
アプリケーションがFeliCa相当以外の場合：ステータスコード0xC0xxを検出することはありません。

2.6.12　ＳＡＭ認証

ＮＦＣモジュールに搭載されているＳＡＭチップに対して認証を行う。
ＳＡＭチップとの通信回数をクリアする。

呼び出し形式：

```
BOOL RFID_SamAuthentication(RFID *hRFID,BYTE mode,LPBYTE datap,
                                                    DWORD data_len, LPDWORD lpStatus);
```

引数：

hRFID	オープン時に取得したハンドル
mode	0x02:（固定値）
datap	認証関連データ（NULL を指定）
data_len	認証関連データ長さ（0 を指定）
lpStatus	ステータスコードへのポインタ、詳細はステータスコード表を参照。

返り値：
FALSE：認証失敗、TRUE: 認証成功

注１）アプリケーションが FeliCa 相当の場合：SAM 通信回数は以下の条件でカウントアップされます。
SAM 通信回数が 65535 回を超えると、SAM との通信ができなくなり、RFID_AutoSelectTarget、RFID_SelectTarget、RFID_Authentication、RFID_Write、RFID_Read で、ステータスコード 0xC0xx を検出します。
その場合、本 API をコールし、SAM 通信回数をクリアして下さい。

API	回数
RFID_AutoSelectTarget	4
RFID_SelectTarget	2
RFID_Authentication	2
RFID_Write	2
RFID_Read	2

注２）下記の場合も SAM 通信回数はクリアされます。
・ＷＢ－１Ｊの起動時、およびＷＢ－１Ｊの再起動時
・RFID_Open をコールした時

注３）FeliCa 相当以外のアプリケーションしか使用しない場合は、本 API を使用する必要はありません。

2.7　ＰＩＣＣ及びアプリケーション固有のデータ構造について

2.7.1　認証パラメータ

１）RFID 認証パラメータ

2.6.9　アプリケーションの認証（RFID_Authentication）をコールする場合、カードのアプリケーションに対応する下記の構造体 RFID_AUTHENTICATION_DATA のメンバにパラメータを設定しデータを引き渡す。

```
typedef struct  {
        RFID_AUTH_STANDARD              AuthStandard;
        RFID_AUTH_FELICA                AuthFelica;
        RFID_AUTH_SSFC                  AuthSSFC;
        RFID_AUTH_MIFARE_CLASSIC        AuthClassic;
        RFID_AUTH_MIFARE_PLUS           AuthPlus;
        RFID_AUTH_MIFARE_ULC            AuthULC;
        RFID_AUTH_DESFIRE               AuthDESFire;
} RFID_AUTHENTICATION_DATA;
```

① ＳＴＡＮＤＡＲＤ

```
typedef struct  {
        WORD            len;
} RFID_AUTH_STANDARD;
```

構造体メンバ

len	認証データサイズ

認証が不要なカードに対して今後の拡張の為にダミーの認証を行う場合は、この構造体を利用する。
※認証データが可変長のアプリケーション（SRIX4K、MIFARE Ultralight 等）については、認証データ用の領域を確保し、確保した領域のサイズを指定する。

② ＦｅｌｉＣａ

```
typedef struct  {
        BYTE            KeyType;
        WORD            SystemCode;
        WORD            SystemKeyVer;
        BYTE            AreaNumber;
        BYTE            AreaCode[64];
        BYTE            ServiceNumber;
        BYTE            ServiceCode[64];
        WORD            GSK_Code;
        WORD            GSK_Ver;
        WORD            USK_Code;
        WORD            USK_Ver;
} RFID_AUTH_FELICA;
```

構造体メンバ

KeyType	認証種別	0x01:エリア／サービス鍵 0x02:ＧＳＫ／ＵＳＫ
SystemCode	システムコード	
SystemKeyVer	システム鍵バージョン（リトルエンディアン）	
AreaNumber	エリア数	
AreaCode	エリアコード／エリア鍵バージョンリスト	
ServiceNumber	サービス数	
ServiceCode	サービスコード／サービス鍵バージョンリスト	
GSK_Code	GSK コード（リトルエンディアン）	
GSK_Ver	GSK バージョン（リトルエンディアン）	
USK_Code	USK コード（リトルエンディアン）	
USK_Ver	USK バージョン（リトルエンディアン）	

ＦｅｌｉＣａに対して認証を行う場合は、この構造体を利用する。

KeyType に 0x01 を指定した場合（エリア/サービス鍵による認証時）は下記のメンバを設定する。
SystemCode, SystemKeyVer, AreaNumber, AreaCode, ServiceNumber, ServiceCode,

KeyType に 0x02 を指定した場合（GSK/USK 鍵による認証時）は下記のメンバを設定する。
SystemCode, GSK_Code, GSK_Ver, USK_Code, USK_Ver

各パラメータに関しての情報はＦｅｌｉＣａの仕様を参照のこと。

③ ＳＳＦＣ

```
typedef struct  {
        WORD            len;
        WORD            SystemKeyVer;
        WORD            AreaKeyVer;
        WORD            ServiceKeyVer;
} RFID_AUTH_SSFC;
```

構造体メンバ

len	認証データサイズ（0x0006）
SystemKeyVer	システム鍵バージョン
AreaKeyVer	エリア鍵バージョン
ServiceKeyVer	サービス鍵バージョン

ＳＳＦＣに対して認証を行う場合は、この構造体を利用する。
※ SSFC は、FeliCa としてアクセスすることも可能。
※ SSFC でも縮退鍵を使用する場合は、FeliCa としてアクセスする。

認証データサイズには 0x0006（SystemKeyVer、AreaKeyVer 及び ServiceKeyVer の合計サイズ）を設定する。
各認証データに関してはＳＳＦＣの仕様を参照のこと。

④ ＭＩＦＡＲＥ　Ｃｌａｓｓｉｃ

```
typedef struct  {
        WORD            len;
        BYTE            fkey[16];
        BYTE            uid[4];
} RFID_AUTH_MIFARE_CLASSIC;
```

構造体メンバ

len	認証データサイズ(0x0014)
fkey	認証データ
uid	UID

ＭＩＦＡＲＥ　Ｃｌａｓｓｉｃに対して認証を行う場合は、この構造体を利用する。

認証データサイズには 0x0014（fkey と uid の合計サイズ）を設定する。
認証データには下記認証データの平文を AES によって暗号化し設定する。
初期ベクトル ALL0、鍵はＮＦＣモジュールの管理者鍵を利用する。

認証データ平文の構造

ofset	意味	size	値
0	鍵種別	1	0x00:KeyA 0x01:KeyB
1	ブロックアドレス	1	0x00～
2	鍵値	6	0xXXXXXXXXXXXX
8	パディングデータ	8	0x8000000000000000

※AES はブロック長 16 バイト、鍵長 16 バイト、CBC モードで暗号化して下さい。

鍵種別、ブロックアドレス、鍵値に関する詳細はＭＩＦＡＲＥ　Ｃｌａｓｓｉｃの仕様を参照のこと。

⑤ ＭＩＦＡＲＥ　Ｐｌｕｓ

```
typedef struct  {
        WORD            len;
        WORD            keyNo;
        WORD            authMode;
        BYTE            fkey[16];
} RFID_AUTH_MIFARE_PLUS;
```

構造体メンバ

len	認証データサイズ(0x0010)	
keyNo	認証鍵番号	
authMode	認証モード	0x0001:First Authenticate
		0x0002:Following Authenticate
fkey	認証データ	

ＭＩＦＡＲＥ　Ｐｌｕｓに対して認証を行う場合は、この構造体を利用する。

認証データサイズには 0x0010（fkey のサイズ）を設定する。
認証データは、認証対象のカードの鍵値（16 バイト）を AES で暗号化し設定する。
初期ベクトル ALL0、鍵はＮＦＣモジュールの管理者鍵を利用する。

認証鍵番号、認証モード、認証データに関する詳細は、ＭＩＦＡＲＥ　Ｐｌｕｓの仕様を参照のこと。

認証データ平文の構造

ofset	意味	size	値
0	鍵値	16	0xXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

※AES はブロック長 16 バイト、鍵長 16 バイト、CBC モードで暗号化して下さい。

⑥ ＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ　Ｃ

```
typedef struct  {
        WORD            len;
        BYTE            fkey[16];
} RFID_AUTH_MIFARE_ULC;
```

構造体メンバ

len	認証データサイズ(0x0010)
fkey	認証データ

ＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ　Ｃの認証時は、このメンバを設定する。
認証データサイズには 0x0010（fkey のサイズ）を設定する。
認証データには認証対象のカードの鍵値を AES により暗号化し設定する。
初期ベクトル ALL0、鍵はＮＦＣモジュールの管理者鍵を利用する。

鍵値の詳細はＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ　Ｃの仕様を参照のこと。

認証データ平文の構造

ofset	意味	size	値
0	鍵値	16	0xXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

※AES はブロック長 16 バイト、鍵長 16 バイト、CBC モードで暗号化して下さい。

⑦ ＭＩＦＡＲＥ　ＤＥＳＦｉｒｅ（ＩＳＯ準拠）

```
typedef struct  {
        WORD            len;
        BYTE            fkey[16];
} RFID_AUTH_MIFARE_ULC;
```

構造体メンバ

len	認証データサイズ(0x0010)
fkey	認証データ

ＭＩＦＡＲＥ　ＤＥＳＦｉｒｅ（ＩＳＯ準拠）に対してダミーの認証を行う場合は、この構造体を利用する。

認証データサイズには 0x0010（fkey のサイズ）を設定する。
認証データには下記の値を暗号化し、設定する。
初期ベクトル ALL0、鍵はＮＦＣモジュールの管理者鍵を利用する。

認証データ平文の構造

ofset	意味	size	値
0	固定データ	16	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

※AES はブロック長 16 バイト、鍵長 16 バイト、CBC モードで暗号化して下さい。

2.7.2　リード／ライトパラメータ

１）RFID リードライトパラメータ

2.6.10　ＰＩＣＣへのライト（RFID_Write）／2.6.11　ＰＩＣＣからのリード（RFID_Read）をコールする場合は、カードのアプリケーションに対応する下記の構造体 RFID_RW_PARAM のメンバにパラメータを設定し、データを引き渡す。

```
typedef struct {
      RFID_RW_STANDARD        RW_Standard;
      RFID_RW_FELICA          RW_Felica;
      RFID_RW_SSFC            RW_SSFC;
      RFID_RW_MIFARE_CLASSIC  RW_CLASSIC;
      RFID_RW_MIFARE_PLUS     RW_PLUS;
      RFID_RW_MIFARE_ULC      RW_ULC;
      RFID_RW_DESFIRE         RW_DESFire;
      RFID_RW_TAG             RW_TAG;
} RFID_RW_PARAM;
```

① ＳＴＡＮＤＡＲＤ

```
typedef struct {
      WORD            pos;
      WORD            len;
} RFID_RW_STANDARD;
```

構造体メンバ

pos	処理開始位置
len	リード/ライトサイズ

ＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ、ＳＲＩX4Kに対してリード/ライトを行う場合は、この構造体を利用する。

リード時にリード/ライトサイズに 0x0000 を指定した場合、全領域の読出しとなる。

ＲＦＩＤ＿Ｒｅａｄ、ＲＦＩＤ＿ＷｒｉｔｅではＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ，ＳＲＩＸ４Ｋのメモリ領域を、0 オリジンの連続アドレス領域としてアクセスする。
処理開始位置、リード/ライトサイズの設定時は注意すること。

カードのアクセス可能領域や、ファイルシステム等はそれぞれのＰＩＣＣの仕様を参照のこと。

② ＦｅｌｉＣａ

```
typedef struct {
      BYTE            CodeFlag;
      BYTE            ServiceNumber;
      WORD            ServiceCodeList[16];
      BYTE            IDt[2];
      BYTE            BlockNumber;
      BYTE            BlockList[36];
} RFID_RW_FELICA;
```

構造体のメンバ

CodeFlag	認証有無 0x00:認証なし 0x01:認証あり
ServiceNumber	サービス数
ServiceCodeList	サービスコードリスト
IDt	RFID_Authentication の返却値を設定
BlockNumber	ブロック数
BlockList	ブロックリスト

ＦｅｌｉＣａに対してリード／ライトを行う場合は、この構造体を利用する。

認証なしのリード/ライトを行う場合は、CodeFlag に 0x00 を設定し、下記のメンバのみを設定する。
ServiceNumber, ServiceCodeList, BlockNumber, BlockList

認証ありのリード/ライトを行う場合は、CodeFlag に 0x01 を設定し、下記のメンバのみを設定する。
BlockNumber, BlockList, IDt

IDt には RFID_Authentication の lpResponseBuffer に格納されたデータを設定すること。

カードのアクセス可能領域や、ファイルシステム等はＦｅｌｉＣａの仕様を参照のこと。

③ ＳＳＦＣ

```
typedef  struct       {
      WORD            pos;
      BYTE            IDt[2];
      WORD            len;
} RFID_RW_SSFC;
```

構造体メンバ

pos	処理対象位置
IDt	RFID_Authentication の返却値を設定
len	リード/ライトサイズ

ＳＳＦＣに対してリード/ライトを行う場合は、この構造体を利用する。

処理対象位置、リード/ライトサイズには下記のデータを設定すること。

処理対象	処理対象位置	リード/ライトサイズ
フラグ領域	0x0001	0x0010
管理領域	0x0002	0x0030
機器伝搬領域	0x0003	0x0030

カードのアクセス可能領域や、ファイルシステム等はＳＳＦＣおよびＦｅｌｉＣａの仕様を参照のこと。

④ ＭＩＦＡＲＥ　Ｃｌａｓｓｉｃ

```
typedef struct {
      WORD            blockNo;
      BYTE            pos;
      BYTE            len;
} RFID_RW_MIFARE_CLASSIC;
```

構造体のメンバ

blockNo	ブロックＮｏ.
pos	処理開始位置
len	リード/ライトサイズ

ＭＩＦＡＲＥ　Ｃｌａｓｓｉｃへのリード／ライトを行う場合は、この構造体を利用する。
リード時にリード/ライトサイズに 0x00 を指定した場合は、16 バイトの読出しとなる。
処理開始位置、リード/ライトサイズ共にブロックサイズを超える指定はできない。

カードのアクセス可能領域や、ファイルシステム等はＭＩＦＡＲＥ　Ｃｌａｓｓｉｃの仕様を参照のこと。

⑤ ＭＩＦＡＲＥ　Ｐｌｕｓ

```
typedef struct {
      WORD            blockNo;
      BYTE            pos;
      BYTE            len;
} RFID_RW_MIFARE_PLUS;
```

構造体のメンバ

blockNo	ブロックＮｏ.
pos	処理開始位置
len	リード/ライトサイズ

ＭＩＦＡＲＥ　Ｐｌｕｓへのリード／ライトを行う場合は、この構造体を利用する。
リード時にリード/ライトサイズに 0x00 を指定した場合は、16 バイトの読出しとなる。
処理開始位置、リード/ライトサイズ共にブロックサイズを超える指定はできない。

カードのアクセス可能領域や、ファイルシステム等はＭＩＦＡＲＥ　Ｐｌｕｓの仕様を参照のこと。

⑥ ＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ　Ｃ

```
typedef struct {
      WORD            pos;
      WORD            len;
} RFID RFID_RW_MIFARE_ULC;
```

構造体メンバ

pos	処理開始位置
len	リード/ライトサイズ

ＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ　Ｃに対してリード/ライトを行う場合は、この構造体を利用する。

処理開始位置に 0x0000 を指定した場合は、全領域の読出しとなる。

ＲＦＩＤ＿Ｒｅａｄ、ＲＦＩＤ＿ＷｒｉｔｅではＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ　Ｃのメモリ領域を、0 オリジンの連続アドレス領域としてアクセスする。
処理開始位置、リード/ライトサイズの設定時は注意すること。

カードのアクセス可能領域や、ファイルシステム等はＭＩＦＡＲＥ　Ｕｌｔｒａｌｉｇｈｔ　Ｃの仕様を参照のこと。

⑦ ＭＩＦＡＲＥ　ＤＥＳＦｉｒｅ（ＩＳＯ準拠）

```
typedef       struct  {
      BYTE            DirName[3];
      BYTE            FileID;
      BYTE            pos[3];
      BYTE            len[3];
} RFID_RW_DESFIRE;
```

構造体メンバ

DirName	ディレクトリ名
FileID	ファイルＩＤ
pos	処理開始位置
len	リード/ライトサイズ

ＭＩＦＡＲＥ　ＤＥＳＦｉｒｅ（ＩＳＯ準拠）に対してリード/ライトを行う場合は、この構造体を利用する。

処理開始位置に全て 0x00 を指定した場合、全領域の読出しとなる。

カードのアクセス可能領域や、ファイルシステム等はＭＩＦＡＲＥ　ＤＥＳＦｉｒｅ（ＩＳＯ準拠）の仕様を参照のこと。

⑧ ＴＡＧ

```
typedef       struct  {
      BYTE            Uid[8];
      BYTE            BlockNo;
} RFID_RW_TAG;
```

構造体メンバ

Uid	ＵＩＤ
BlockNo	ブロックＮｏ.

ＴＡＧに対してのリード/ライトを行う場合は、この構造体を利用する。

カードのアクセス可能領域や、ファイルシステム等に関しては ISO15693（TAG）の仕様を参照のこと。

2.8　ステータスコード表

Sーソフトウェア要因（パラメータの設定ミスなど)、Hーハードウェア要因（故障など）

1）API ステータス

上位	下位	意　　味	要因
0x90	0x00	正常終了	S
0x20	0x00	API コールの引数が正しくない	S
	0x01	無効なハンドルが指定された	S
	0x02	十分なメモリが確保できない	S
	0x03	API コールシーケンスが正しくない	S
	0x05	レスポンス受信タイムアウト	S、H
	0x06	不正なエリアアクセス	S

2）WB-1J～RFID モジュール間ステータス

上位	下位	意　　味	要因
0x10	0x01	PARITY　ERROR	H
	0x02	FRAMING ERROR	H
	0x04	OVERRUN ERROR	H
	0x08	CRC ERROR	H
	0x10	ブロック間通信タイムアウト	H
	0x20	キャラクタ間通信タイムアウト	H

３）NFC 対応モジュールのステータス

上位	下位	意　　味	要因
0x90	0x00	正常終了	
0x63	0x00	製造者鍵、管理者鍵の照合不一致	S
	0xCX	製造者鍵、管理者鍵の照合不一致［X によって残りの試行回数を示す］	S
0x64	0x00	書き込みエラー[実行エラー]	S、H
0x65	0x81	書き込みエラー（メモリは書き換わった可能性あり）[実行エラー]	S、H
0x67	0x00	レングス異常	S
0x69	0xXX	コマンドとしての記述は正しいが、他の要因でコマンドが許可されない	S
	0x84	閉塞中	H
	0x85	コマンドの使用条件が満たされない	S
0x6A	0xXX	Value フィールドの Data の値による異常	S
	0x80	value フィールドの Data の値が正しくない	S
	0x81	value フィールドの Data で指定された機能はサポートされていない	S
	0x82	アクセス対象ファイルが無い	H
	0x84	メモリが足りない	S
	0x86	パラメータの値が正しくない	S
	0x88	参照された鍵が正しく設定されていない	S
0x6B	0x00	RF 通信対象のメモリの範囲外のオフセットを指定している	S
0x6D	0x00	value フィールドの Instruction の値がサポートしていないものを指定している	S
0x6E	0x00	コマンドフォーマット異常	H
0x6F	0xXX	自己診断異常	H
0xC0	0xXX	暗号モジュール異常	S、H
0xE0	0xXX	その他異常	H
	0x80	Host I/F より受信したデータがパリティーエラー、フレーミングエラー、オーバランエラー	H
	0x81	タイムアウト	S、H
	0x82	無効なデータを受信	H
	0x83	致命的なエラー	S、H
	0x84	磁界が供給されていない	S、H
	0x85	RF 通信の CRC エラー	H

4）ISO15693 対応モジュールのステータス

上位	下位	意　　味	要因
0x90	0x00	正常終了	
0x00	0x10	モジュールが送信に失敗	H
	0x11	RFID タグから応答なし	S、H
	0x12	モジュールが受信エラーを検出	H
	0x13	モジュールが衝突エラーを検出	S、H
	0x20	データ長が不正	H
	0x21	Function_Code が未定義	H
	0x22	Protocol_Code が未定義	H
	0x23	データ内容が不正	S
	0x40	RFID タグからエラー応答を受信	S、H
	0x41	Tag 複数検出	H
	0x6x	プロトコル異常	H
	0x7x	RFID タグからエラー応答を受信	H