3DS

3DS プログラミングマニュアル

NFP ライブラリ編

2016-05-10

Version 1.4

目次

コード

表

図

1. はじめに

本ドキュメントは、NFP(Nintendo Figurine Platform / NFC Featured Platfrom の略)をアプリケーションで利用する開発者を対象に、NFP ライブラリの概要、利用する関数、プログラミング手順などについて説明したものです。

アプリケーションで NFP を利用すると、amiibo™(アミーボ)と呼ばれるキャラクター商品とアプリケーションを連携させることができるようになります。

なお、本ドキュメントでは NFP および amiibo についての説明を割愛していますので、事前に「NFP オーバービュー」および「企画・運用マニュアル – NFP 編」を一読しておくことを推奨します。

また、NFP ライブラリから特定のエラーが返ってきた場合に表示するメッセージ例の情報を、別途エラーコードリストに記述しています。必要に応じて参照することを推奨します。

注意: CTR では、赤外線通信を用いて NFC リーダー/ライターと通信します。
CTR でアプリケーションから NFP ライブラリを使用する際、赤外線通信を利用する他の機能を使用中ならば、その機能を先に終了させてください。

赤外線通信は以下の機能で利用されています。

- 本体間赤外線通信
- 拡張スライドパッド

2. NFP ライブラリでできること

NFP ライブラリは、CTR 向けの NFC リーダー/ライター、または SNAKE 本体に搭載された NFC（Near Field Communication）モジュールを介して、NFP タグ（NFP 用にフォーマットされた IC タグ）の読み書きを行うためのライブラリです。

NFP ライブラリでは現在、以下の機能に対応しています。

表 2-1. NFP ライブラリが対応している機能

機能	備考
NFC タグの検知	NFC リーダー/ライターでは amiibo を、SNAKE では amiibo および交通系 IC カード（Suica, PASMO, Kitaca, TOICA, manaca, ICOCA, SUGOCA, nimoca, はやかけん）を検知できます。なお本ドキュメントでは、検知可能な IC タグや IC カードを総称して NFC タグと記載しています。 上記に記載していない NFC タグも検知できる場合がありますが、NFP ライブラリでは動作を保証していません。
NFP タグの読み書き	読み書きが可能なのは NFP タグのみです。
NFP タグのバックアップ	NFP ライブラリを使用して NFP タグにデータを書き込むと、自動的にライブラリがそのデータを本体（「amiibo設定」のセーブデータ）にバックアップします。 NFP タグ内のデータが破損していた場合、アプリケーションからの明示的な操作によって、バックアップしておいたデータから NFP タグ内のデータを復旧することができます。

また、タグは同時に一つしか使えません。タグを同時に二つ以上を認識させようとすることはサポート対象外となり、正常に動作しません。NFC リーダー/ライター使用時に、このような使用方法をした場合、NFC リーダー/ライターがハングアップしてしまうことがあり、電源を入れ直さなければならない状態になります。

3. 共用領域とアプリ専用領域

NFP タグ内には、共用領域とアプリ専用領域の 2 つのデータ領域が存在します。

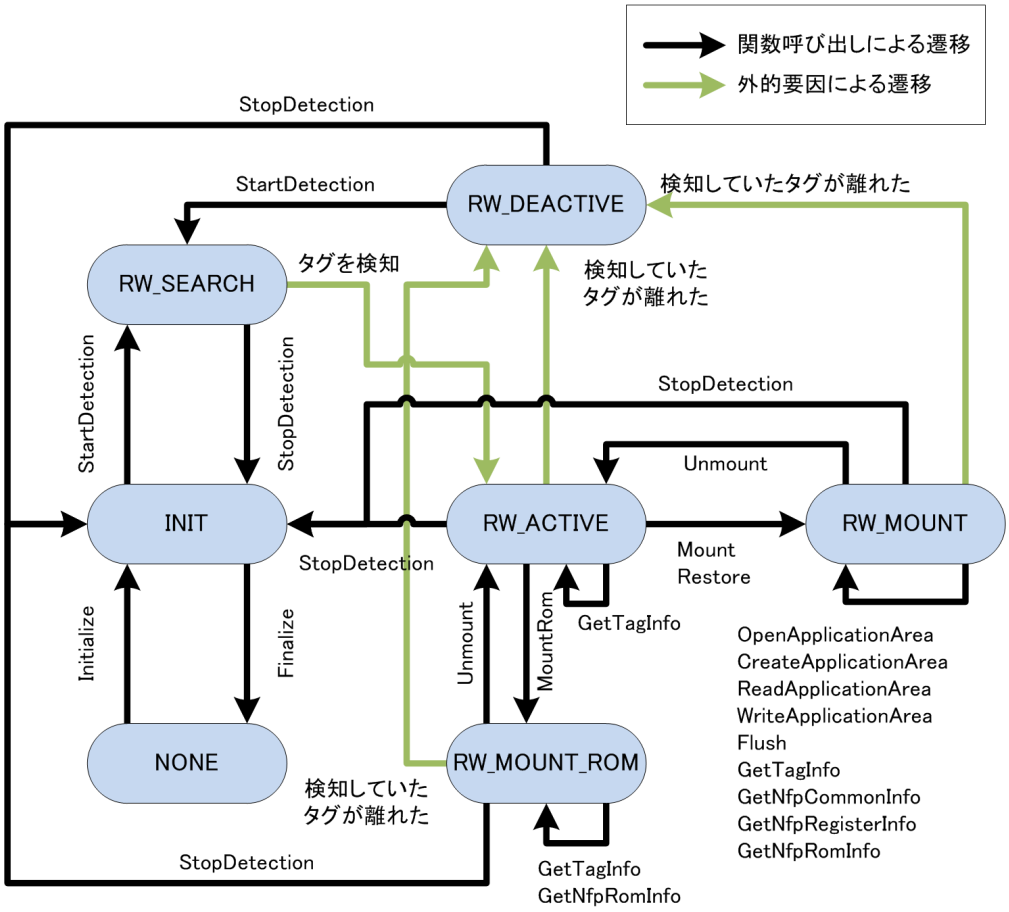
共用領域には、キャラクター ID といった amiibo 自体の情報と初期登録時の情報が格納されています。共用領域は、NFP 対応アプリケーションであれば読み込むことができますが、「amiibo設定」(システムで用意されたアプリケーション)でなければ書き込むことができません。

アプリ専用領域には、最大 216 バイトのデータを格納することができます。格納するデータはアプリケーションの自由ですが、複数のアプリケーションのデータを同時に格納しておくことはできません。なお、アプリ専用領域にデータが格納されていた場合、そのアプリ専用領域を作成したときのアクセスID と同じアクセスID でなければ読み書きができません。アプリ専用領域のデータを上書きする場合は、「amiibo設定」でアプリ専用領域のデータを消してから書き込むようにユーザーを誘導してください。

4. ステート遷移

NFP ライブラリのステートは、以下のように遷移します。

図 4-1. NFP ライブラリのステート遷移図



注意: 図には書かれていませんが、各ステートで `nn::nfp::Finalize()` を呼び出すと `NONE` に遷移します。また、`NONE` と `INIT` を除くステートに遷移している状態で無線オフモードまたはスリープモードに入るとステートが `INIT` に遷移します。

注意: CTR 向けの NFC リーダー/ライター 使用時に、NONE, INIT 以外のステートで NFC リーダー/ライター との接続が切断されると、INIT に遷移します。

ステートは `nn::nfp::NfpState` で定義されており、`nn::nfp::GetNfpState()` で現在のステートを取得することができます。

各ステートは以下のような状態を示しています。

表 4-1. ステートが示す状態

ステート	状態の説明
NONE	NFP ライブラリを初期化する前の状態です。
INIT	NFP ライブラリの初期化は完了しているが、まだタグの探索を行っていない状態です。
RW_SEARCH	タグの探索を行っている状態です。
RW_ACTIVE	タグを検知した状態です。
RW_DEACTIVE	検知していたタグが通信可能範囲から離れた状態です。
RW_MOUNT	検知していたタグをマウントした状態です。検知していたタグが NFP タグでなければマウントできません。
RW_MOUNT_ROM	検知していたタグをマウントした状態です。RW_MOUNT との違いは、アクセス可能な情報が制限されていることです。

表 4-2. 各ステートで使用可能な関数

ステート	使用可能な関数（名前空間 nn::nfp は省略しています）
NONE	`Initialize()`
INIT	`SetActivateEvent(), SetDeactivateEvent(), StartDetection()`
RW_SEARCH	`StopDetection()`
RW_ACTIVE	`GetTagInfo(), Mount(), MountRom(), Restore(), StopDetection()`
RW_DEACTIVE	`StartDetection(), StopDetection()`

RW_MOUNT	`CreateApplicationArea(),`
	`Flush(), GetNfpCommonInfo(), GetNfpRegisterInfo(), GetNfpRomInfo(), GetTagInfo(),`
	`OpenApplicationArea(), ReadApplicationArea(), StopDetection(), Unmount(), WriteApplicationArea()`

RW_MOUNT_ROM	`GetNfpRomInfo(), GetTagInfo(),` `StopDetection(), Unmount()`
NONE 以外	`Finalize(), GetConnectionStatus(), InitializeCreateInfo()`
すべて	`GetNfpState()`

NFP ライブラリを使用することで、SNAKE の消費電力が変動し、バッテリ持続時間に影響を与えます。NFP ライブラリのステートと、消費電力の関係（バッテリ持続時間への影響）は以下の通りです。

ステート	SNAKE のバッテリ持続時間
NONE	変化なし
INIT, RW_SEARCH, RW_DEACTIVE	1 % 未満の減少
RW_ACTIVE, RW_MOUNT, RW_MOUNT_ROM	約 8 % の減少

NFC リーダー/ライターのバッテリ持続時間についての情報は、以下の通りです。CTR 側のバッテリ持続時間は、赤外線通信使用時と同じになります。

ステート	NFC リーダー/ライター との 接続状態	NFC リーダー/ライター の バッテリ持続時間
NONE, INIT	切断	変化なし
INIT, RW_DEACTIVE	接続	約 40% の減少
RW_SEARCH	接続	約 94% の減少
RW_ACTIVE, RW_MOUNT, RW_MOUNT_ROM	接続	約 65 % の減少

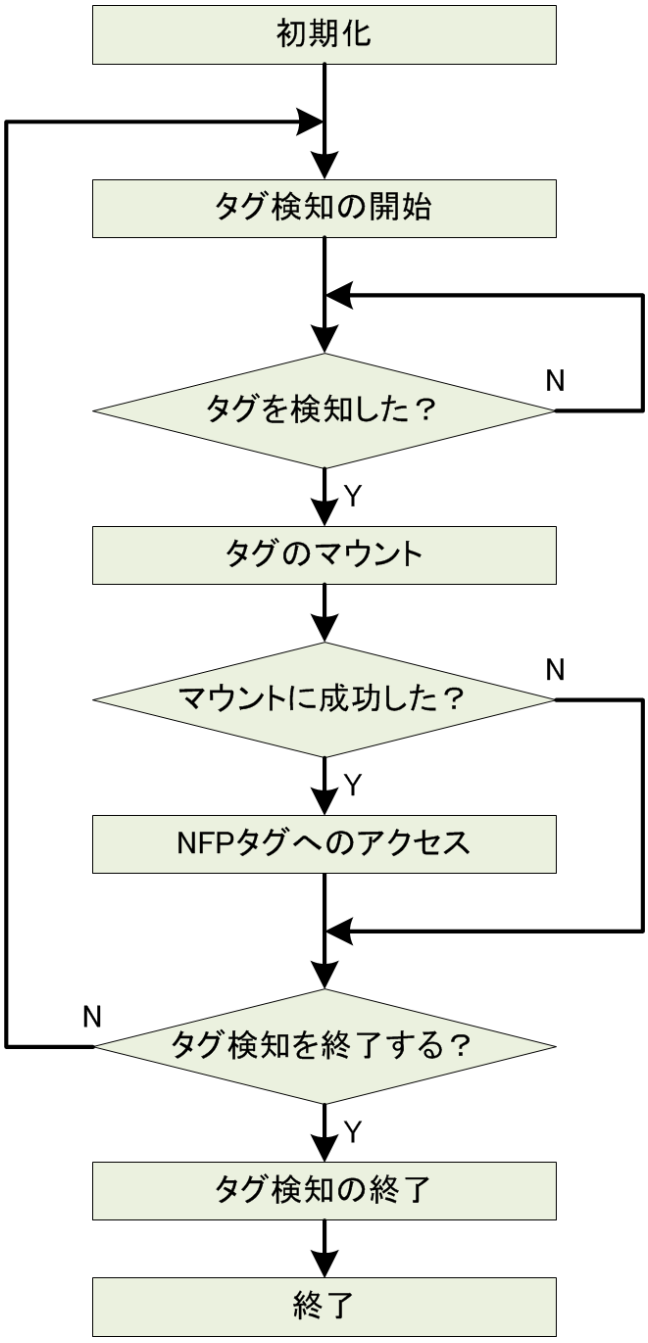
可能な範囲で、消費電力を抑えられるステートに遷移することを推奨します。ただし、`nn::nfp::Initialize()` は処理に 1 秒程度の時間を要するため、過剰な頻度で `nn::nfp::Finalize()` を呼ぶ必要はありません。

5. 処理フロー

この章では、NFP ライブラリを利用する際の処理フローを、全体図と各処理を細分化したシーケンスで説明します。

NFP ライブラリを利用する際の処理フローの全体図を簡単に表すと以下のようになります。

図 5-1. NFP ライブラリを利用する際の処理フロー（全体図）

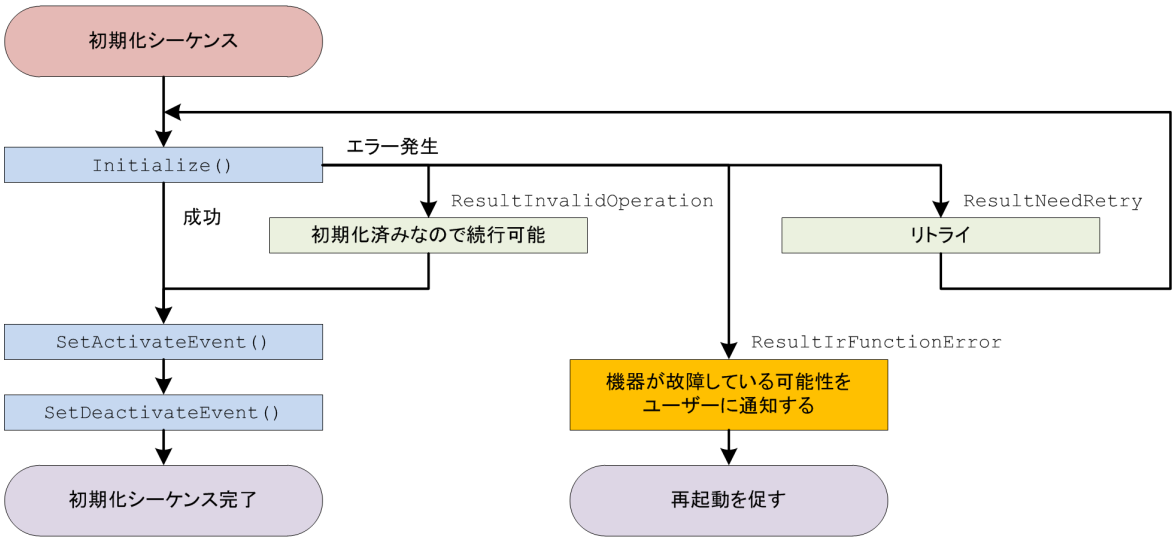


5.1. 初期化シーケンス

NFP ライブラリの初期化を行うシーケンスです。

アプリケーションの途中で NFP ライブラリの終了処理を行わない限り、アプリケーションの起動から NFP ライブラリの関数を呼び出すまでに、一度だけ行います。

図 5-2. 初期化シーケンスのフロー図

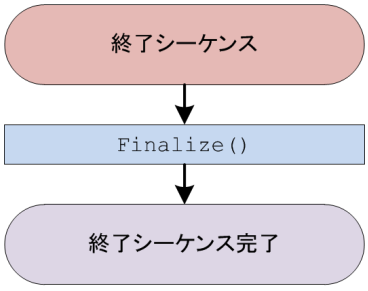


5.2. 終了シーケンス

NFP ライブラリの終了処理を行うシーケンスです。

アプリケーションで NFP ライブラリを使用する必要がなくなったときに行います。

図 5-3. 終了シーケンスのフロー図



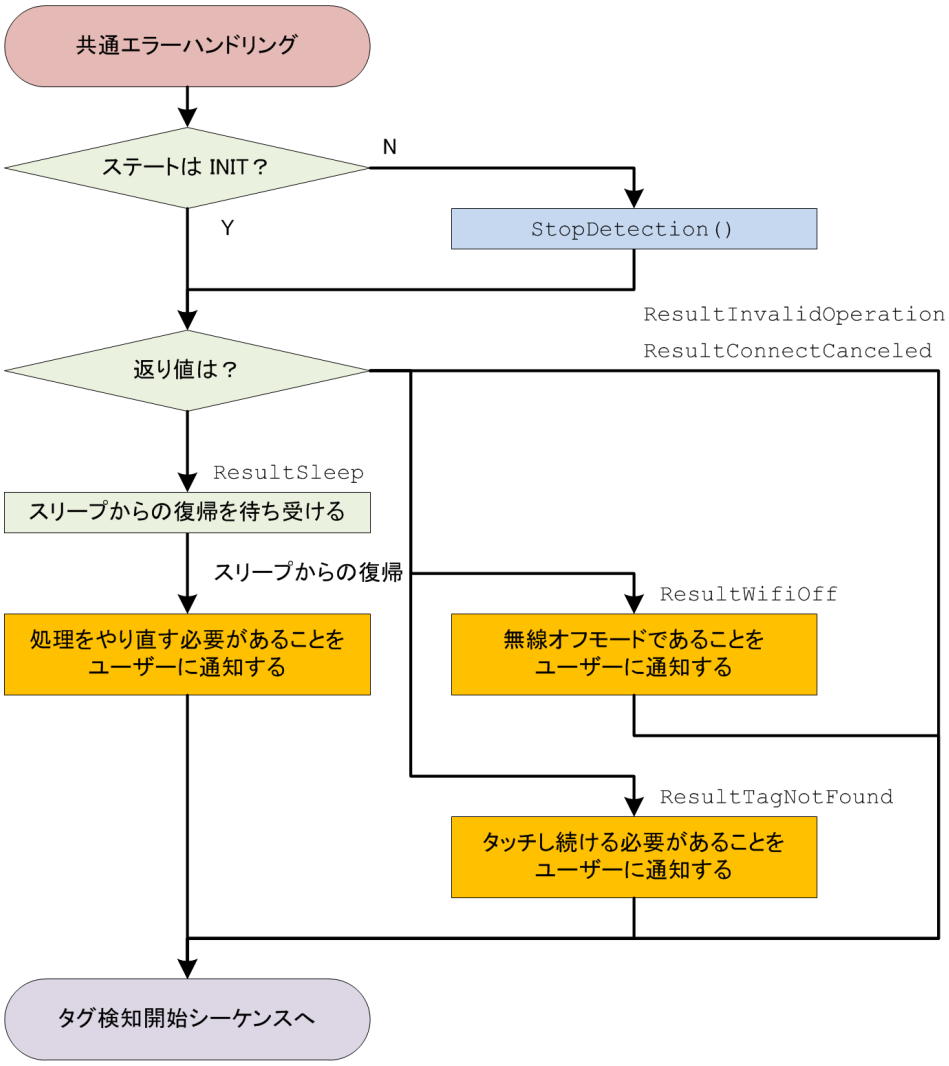
5.3. 共通エラーハンドリング

NFP ライブラリの関数を呼び出した際に、共通して返されるエラーをハンドリングするシーケンスの例です。

このシーケンス例でハンドリングしているエラーは、基本的にタグの検知開始からやり直す必要のあるエラーです。また、通信が完了するまで同じタグでタッチし続け、別のタグに入れ替わったときにはエラーが発生することを前提にしています。

無線オフモードであれば無線オンモードにすることでそのエラーが発生しなくなるように、エラーの原因をユーザーの手で取り除く方法がある場合は、ユーザーにその方法を通知してください。

図 5-4. 共通エラーハンドリングのフロー図

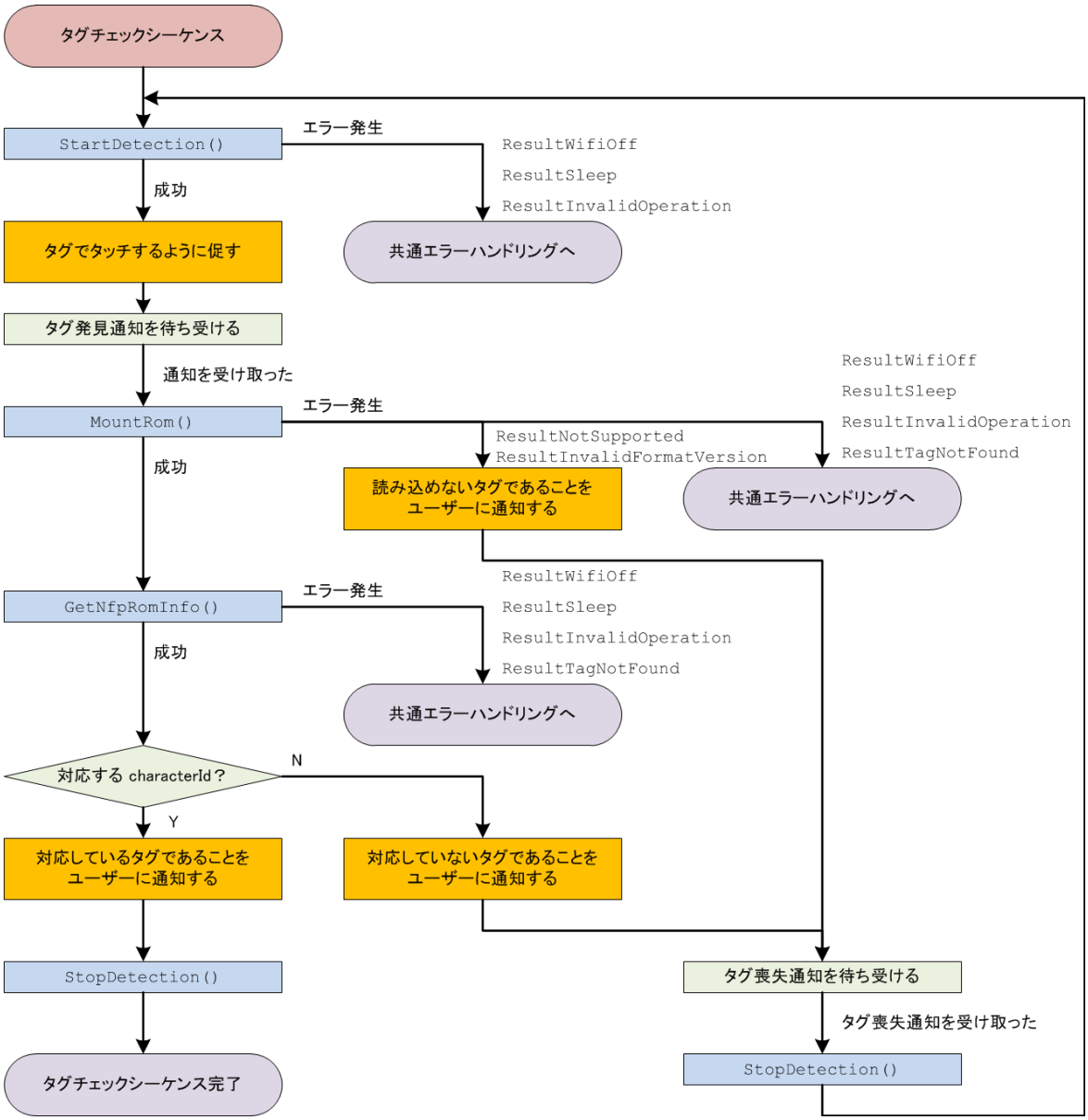


補足： CTR 向けの NFC リーダー/ライターを利用する場合は、タグ検知開始シーケンスではなく、NFC リーダー/ライター接続シーケンスに進むようにしてください。

5.4. タグチェックシーケンス

オーナーの情報やアプリ専用領域の情報にはアクセスしないアプリケーションで、どのキャラクターの NFP タグがタッチされたかをチェックするシーケンスです。

図 5-5. タグチェックシーケンスのフロー図

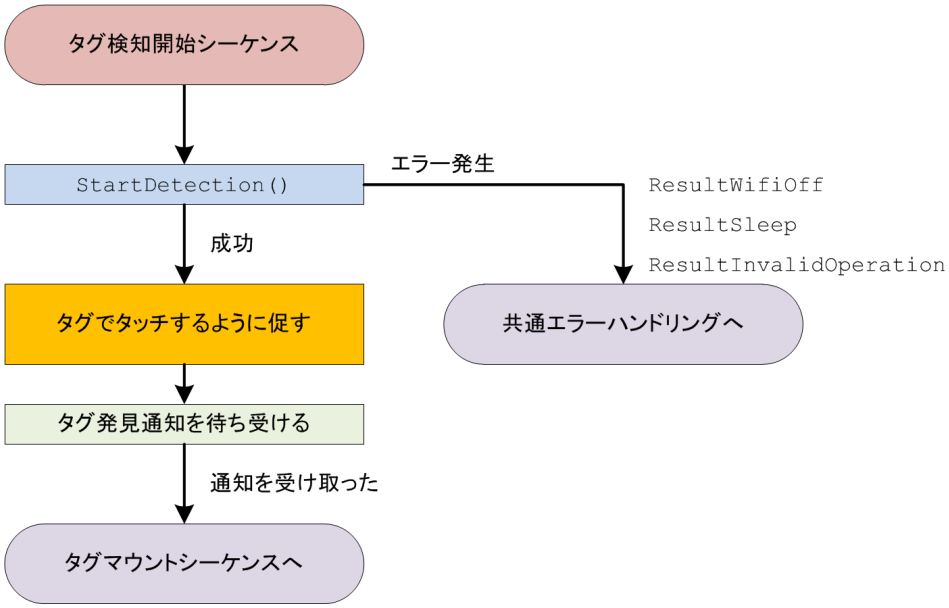


5.5. タグ検知開始シーケンス

NFC モジュールによるタグの探索を開始し、ユーザーにタグのタッチを促すシーケンスです。

フローにはありませんが、必ずタグ発見通知の待ち受けをユーザーがキャンセルできるようにしてください。

図 5-6. タグ検知開始シーケンスのフロー図

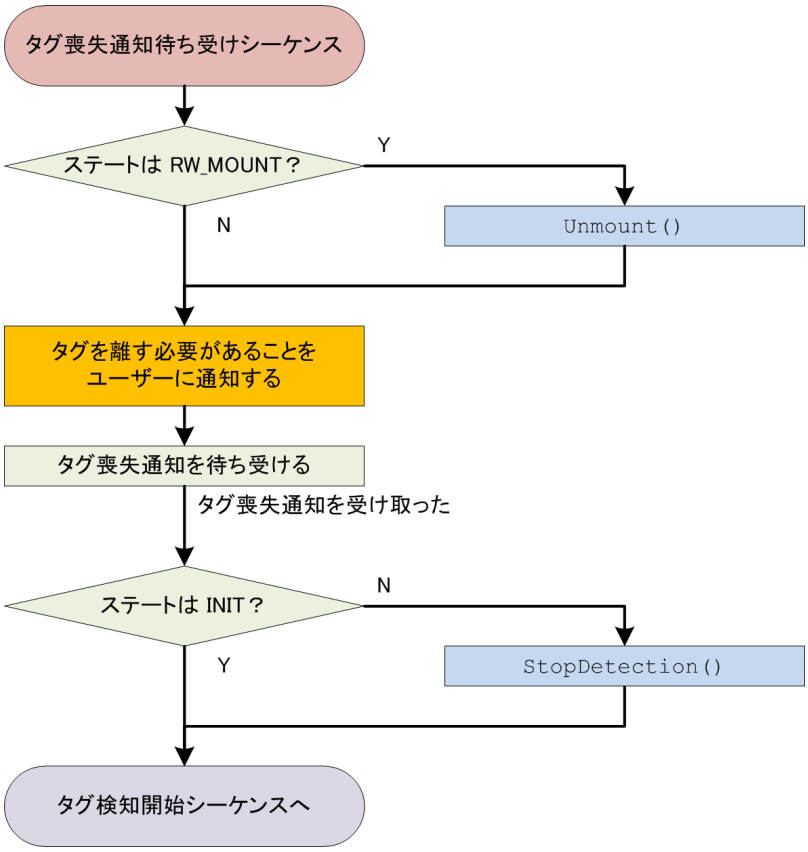


5.6. タグ喪失通知待ち受けシーケンス

タグとの通信が完了したときや、処理対象のタグとは別のタグが検知されたときなど、タッチしているタグを NFC モジュールの通信可能範囲内から離す必要がある場合に行われるシーケンスです。

フロー図では引き続き別のタグを検知することを想定していますが、通信完了後にタグの検知を行わない場合はタグ喪失通知を待ち受ける必要はありません。

図 5-7. タグ喪失通知待ち受けシーケンスのフロー図



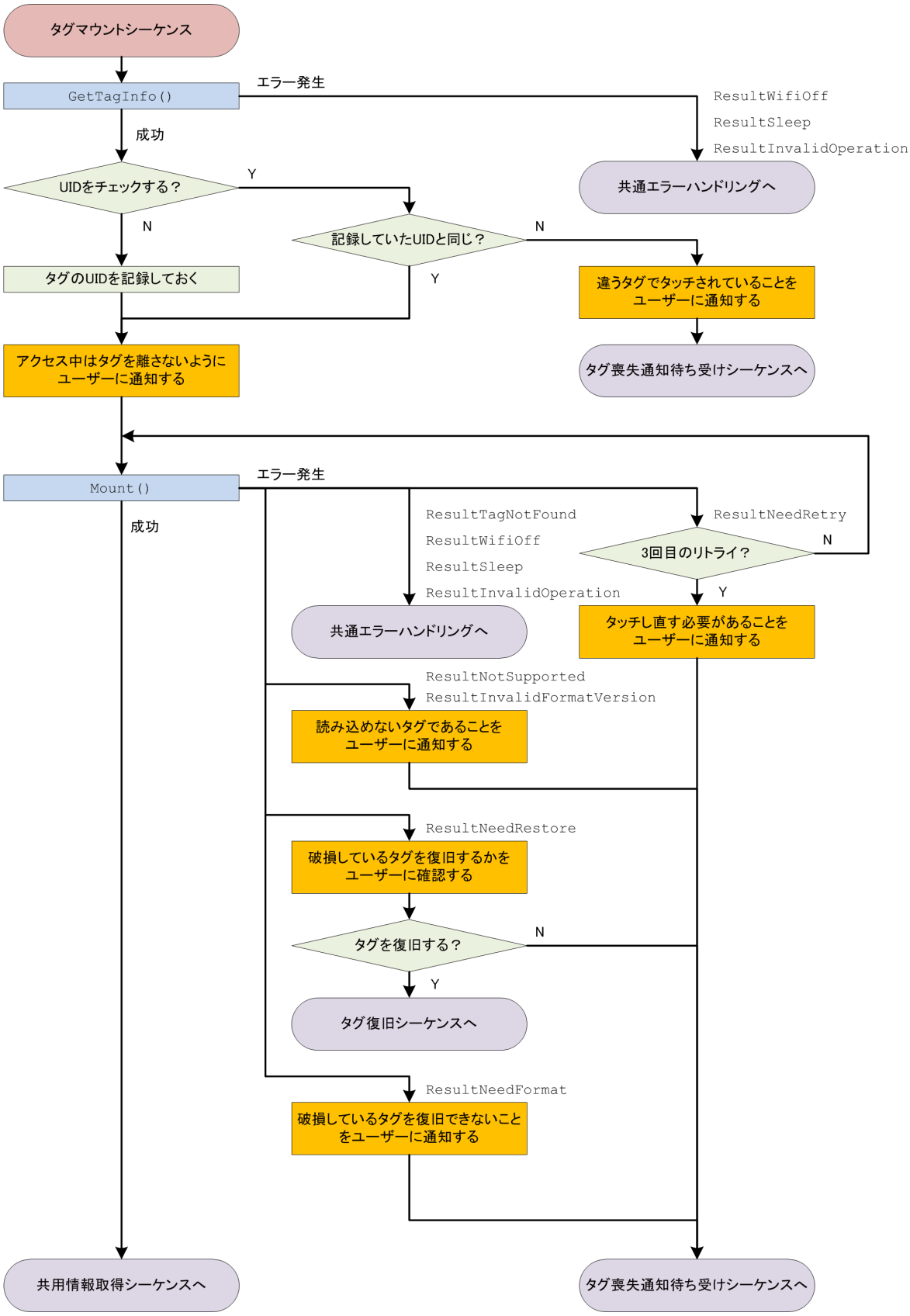
5.7. タグマウントシーケンス

NFP ライブラリが検知したタグをマウントするシーケンスです。

フロー図では、エラーのために処理を再開する必要があった場合や前のシーンで読み込んでいたタグに書き込みを行う場合を想定し、マウントを行う前にタグ情報の取得を行って、タグの UID をチェックするようにしています。

処理でエラーが発生してもリトライが可能な場合の再試行数はフロー図で示した回数でなくてもかまいません。

図 5-8. タグマウントシーケンスのフロー図

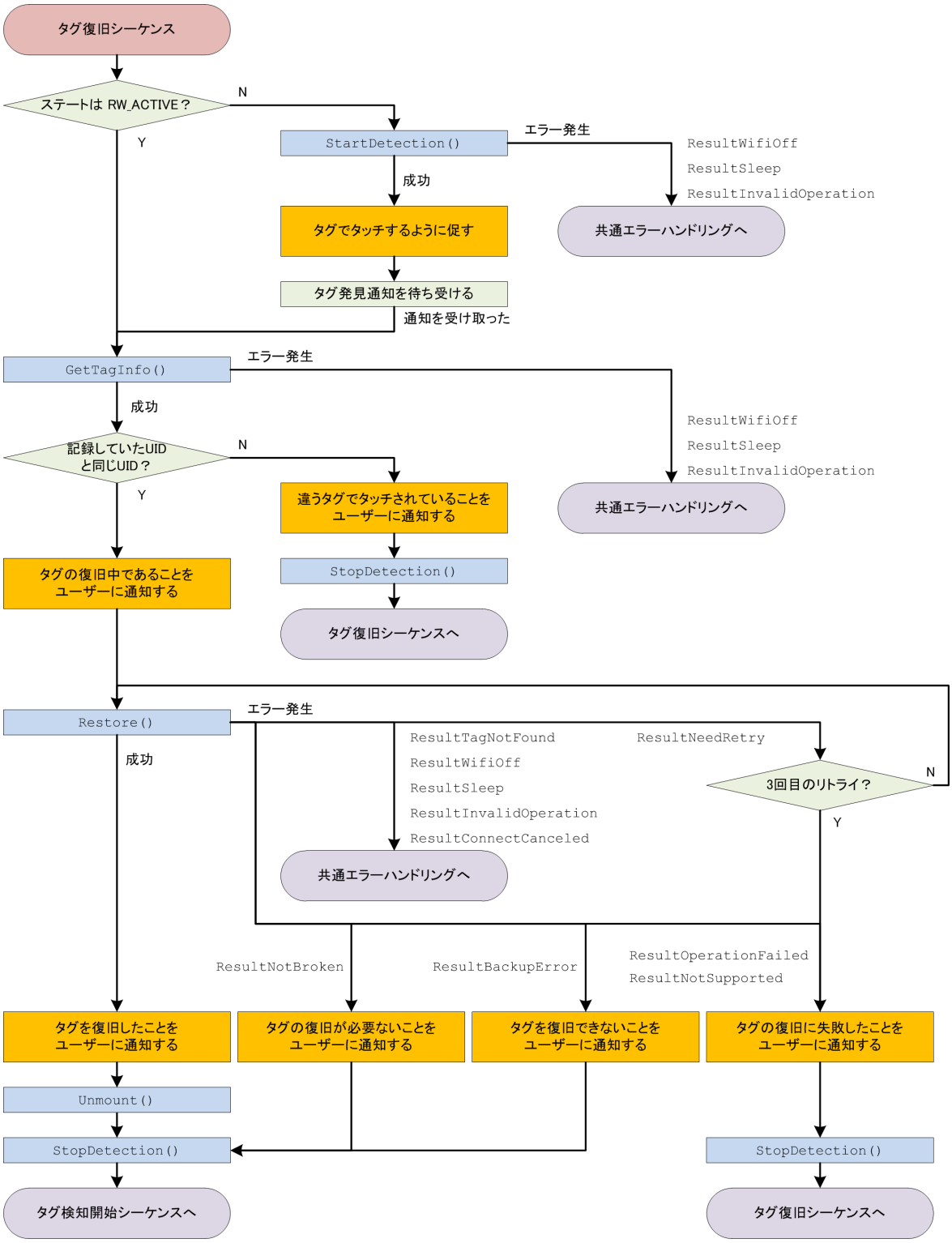


5.8. タグ復旧シーケンス

NFP タグが破損していたものの、アプリケーションで復旧可能な場合にタグを復旧するシーケンスです。以下のシーケンスを実装せず、amiibo 設定を復旧モードで起動することでも NFP タグを復旧できます。

フロー図では、マウントを試みたあとにタグが離された場合や違うタグがタッチされた場合に備えて、タグの検知から再開するようにしています。

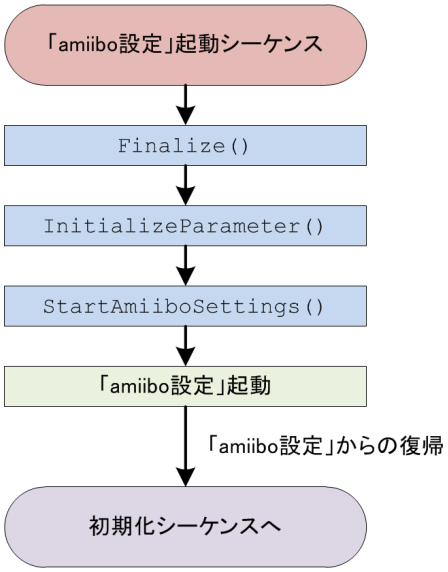
図 5-9. タグ復旧シーケンスのフロー図



5.9. 「amiibo設定」起動シーケンス

NFP タグの初期化やアプリ専用領域の削除などを行うために、「amiibo設定」を起動するシーケンスです。

図 5-10. 「amiibo設定」起動シーケンスのフロー図



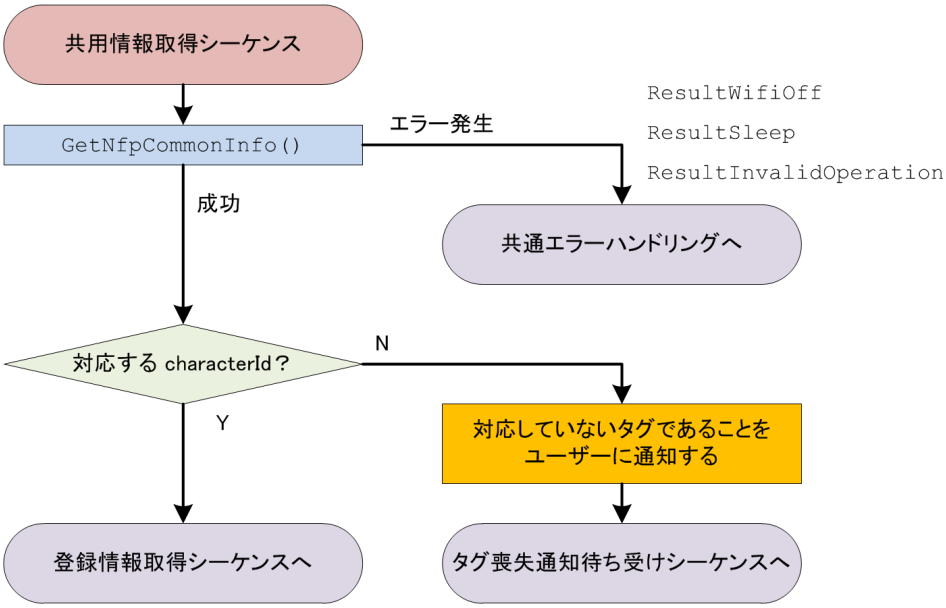
5.10. NFP タグへのアクセスを行う処理フロー

5.10.1. 共用情報取得シーケンス

NFP タグの共用領域に記録されている共用情報を取得するシーケンスです。

アプリケーションが対応するキャラクターの NFP タグであるかどうかを判定し、共用情報に記録されているキャラクターID（characterId）がアプリケーションで対応していないものである場合は、タッチされた NFP タグには対応していないことをユーザーに通知してください。

図 5-11. 共用情報取得シーケンスのフロー図



5.10.2. 登録情報取得シーケンス

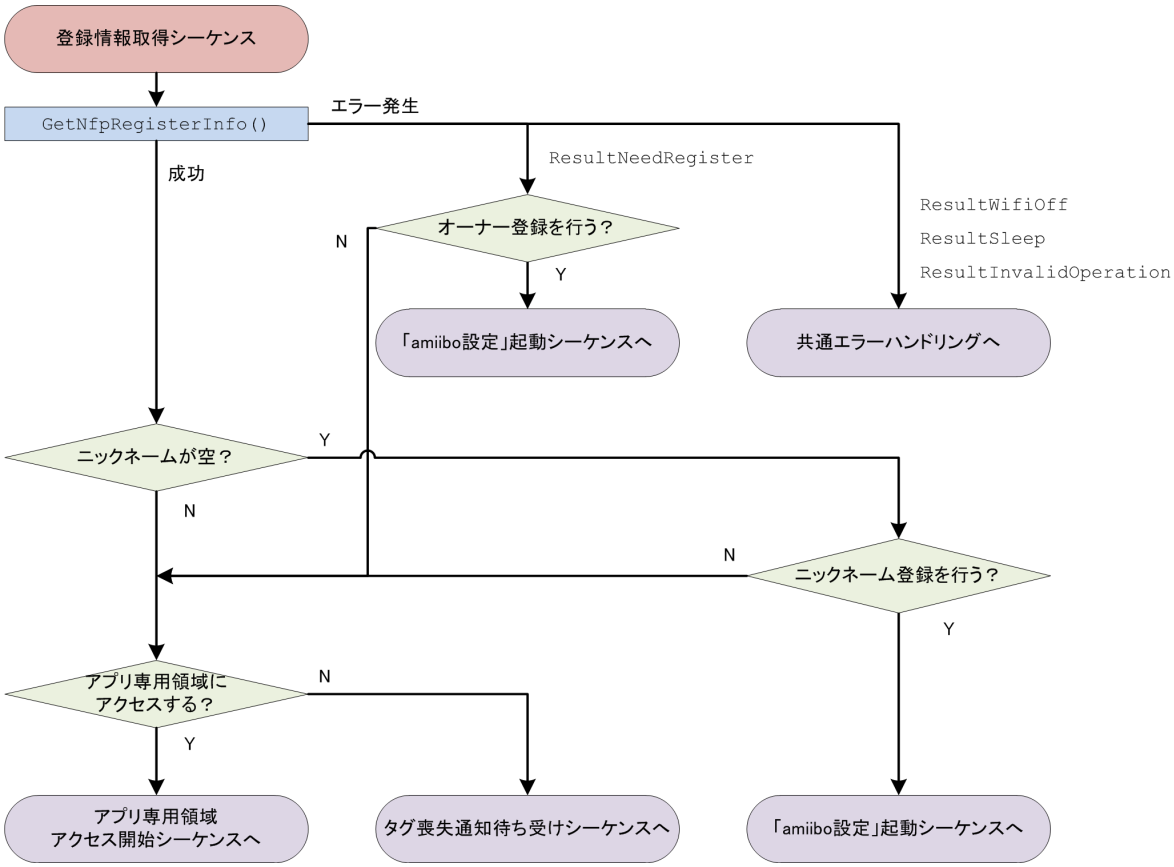
NFP タグの共用領域に記録されている登録情報を取得するシーケンスです。

登録情報には、オーナーの Mii のデータや amiibo のニックネームが記録されています。

アプリケーションがオーナーの Mii のデータを必要とする場合、オーナー登録されていない NFP タグであることを検知したときには、「amiibo設定」でオーナー登録をするために、「amiibo設定」を起動するシーケンスに移ってください。

アプリケーションが amiibo のニックネームを必要とする場合、ニックネームが空の NFP タグであることを検知したときには、「amiibo設定」でニックネームを登録をするために、「amiibo設定」を起動するシーケンスに移ってください。

図 5-12. 登録情報取得シーケンスのフロー図



5.10.3. アプリ専用領域アクセス開始シーケンス

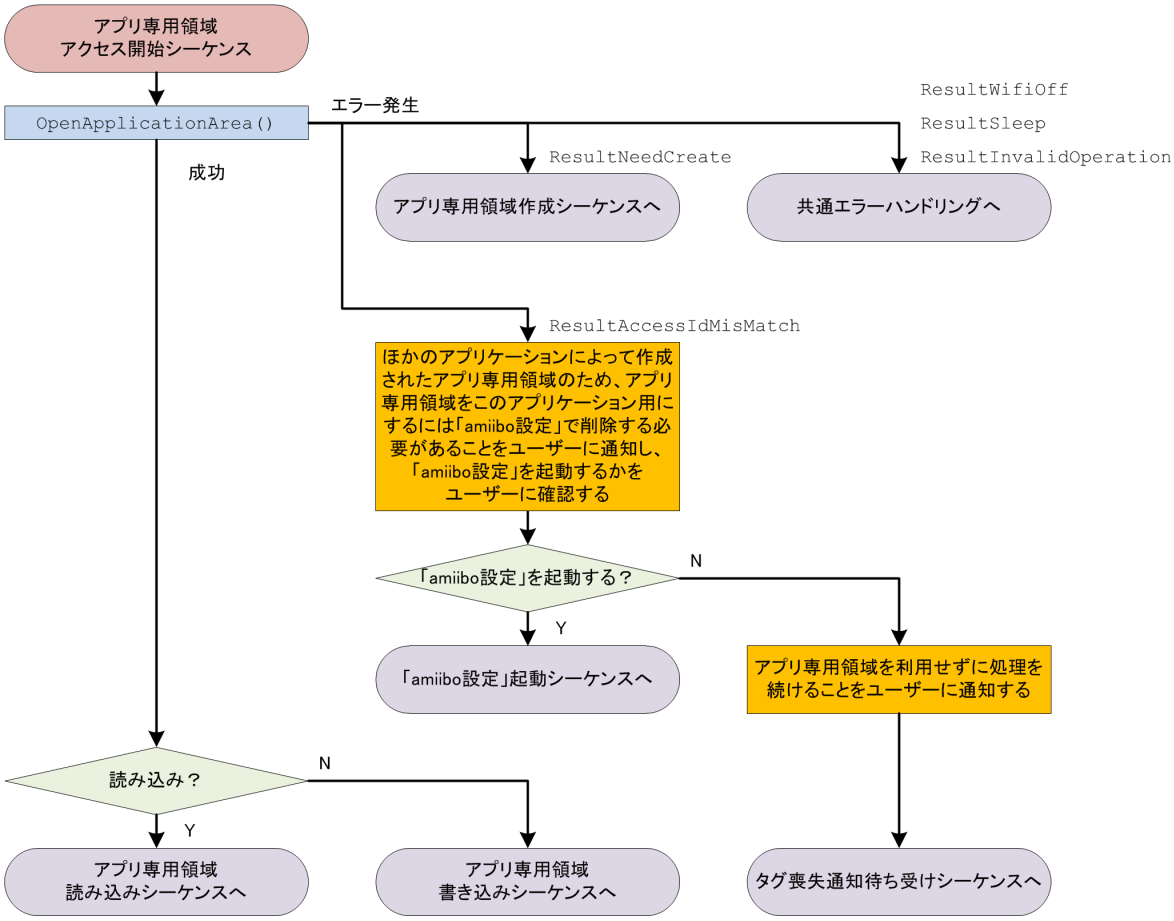
マウントした NFP タグのアプリ専用領域に対するアクセスを開始するシーケンスです。

アプリ専用領域が NFP タグ内に作成されていない場合は、アプリ専用領域を作成するためにアプリ専用領域作成シーケンスへと遷移してください。

すでにアプリ専用領域が作成されていて、アプリ専用領域が作成されたときに指定されたアクセスID がアプリケーションのアクセスID と異なる場合は、そのままではアプリ専用領域の読み書きを行うことができません。「amiibo設定」でアプリ専用領域を削除する必要がありますので、ユーザーにその旨を通知して、「amiibo設定」を起動してください。

上記の通知を行うときには、アプリケーションがアプリ専用領域を利用した場合と利用しない場合の挙動の違いをユーザーに通知することを推奨します。また、アプリ専用領域を利用しなくてもゲームを続けることができるようにしてください。

図 5-13. アプリ専用領域アクセス開始シーケンスのフロー図



5.10.4. アプリ専用領域作成シーケンス

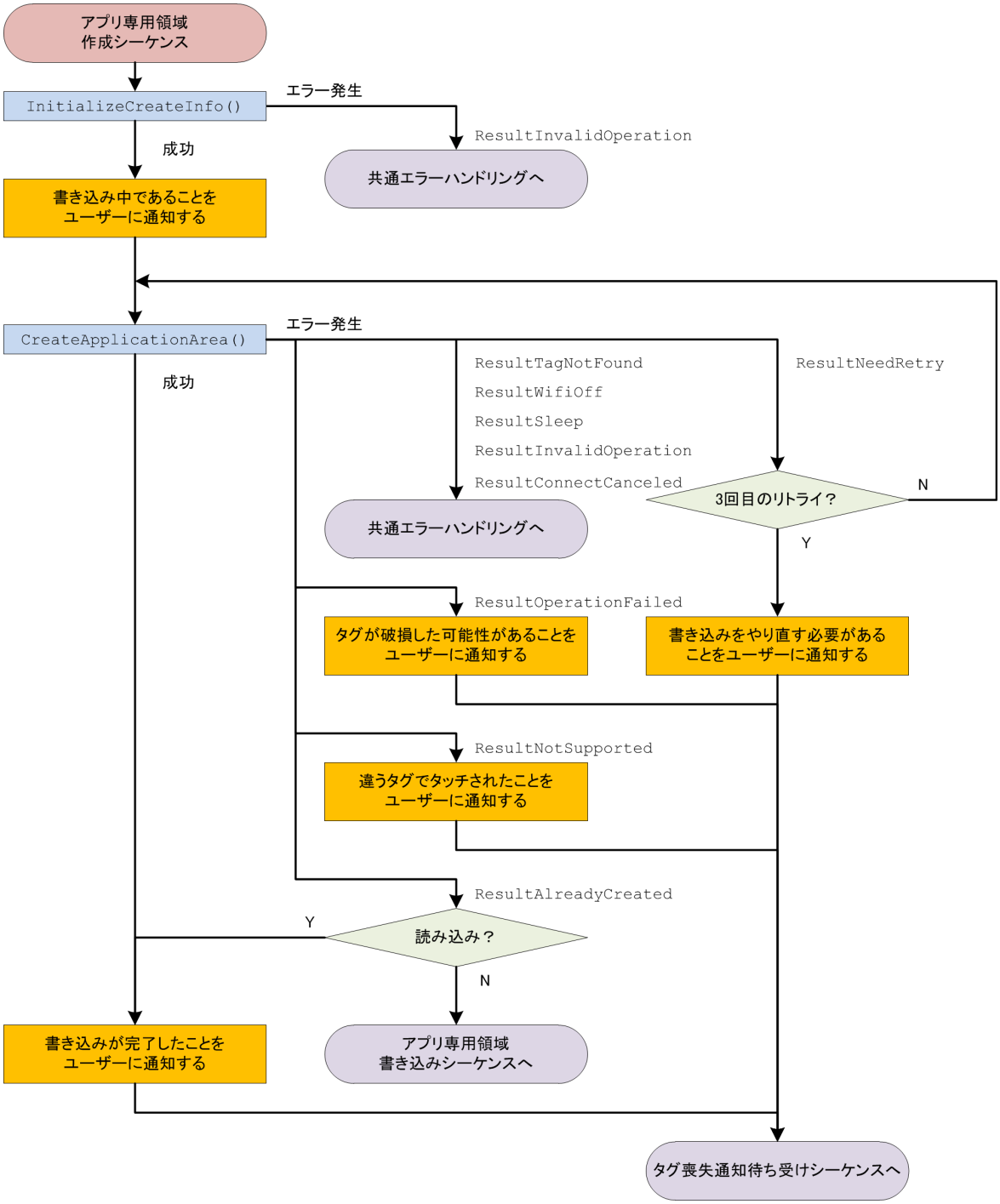
アプリ専用領域を作成するシーケンスです。

アプリ専用領域にデータを書き込むためにアプリ専用領域へのアクセスを開始した際にこのシーケンスへと遷移していた場合は、書き込もうとしていたデータを `nn::nfp::InitializeCreateInfo()` に渡す初期化データとすることで `nn::nfp::CreateApplicationArea()` が成功した時点で書き込みのシーケンスに遷移する必要はなくなります。

作成済みであることを `nn::nfp::CreateApplicationArea()` が返した場合はアプリ専用領域の内容が意図したものではない可能性があります。

処理でエラーが発生してもリトライが可能な場合の再試行数はフロー図で示した回数でなくてもかまいません。

図 5-14. アプリ専用領域作成シーケンスのフロー図



5.10.5. アプリ専用領域チェックシーケンス

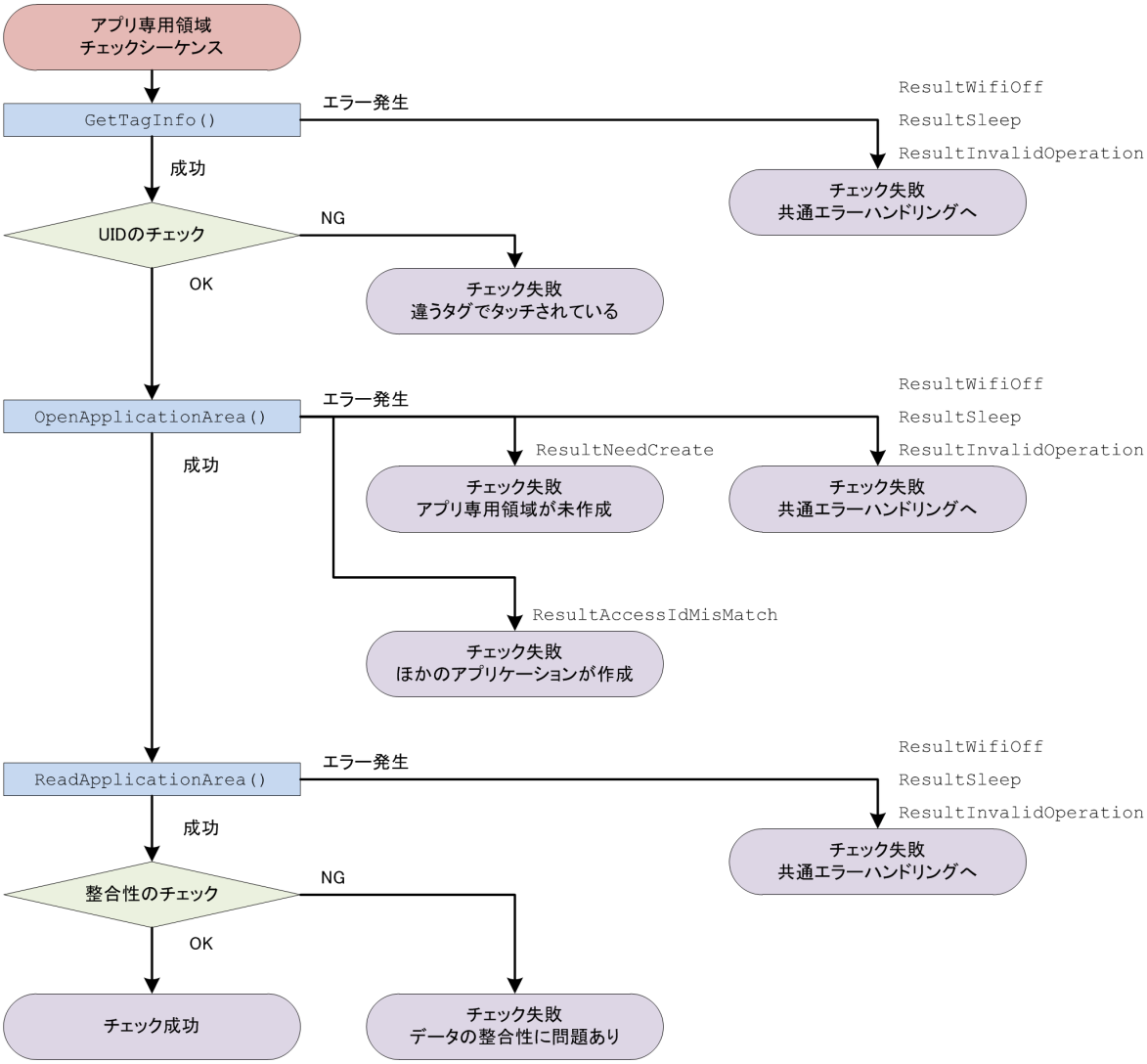
書き込みを行う直前などにアプリ専用領域の整合性を確認するシーケンスです。

ゲームシーンの前後でアプリ専用領域の読み込みと書き込みを行うなど、アプリ専用領域の読み込みと書き込みとの間に時間がある場合は、以下のような行為で書き込みを行えなくなることがあるため、書き込みを行う前にアプリ専用領域の整合性を確認することを推奨します。

- 別の NFP タグでタッチする
- ほかの機器でアプリ専用領域を書き換える
- 「amiibo設定」でアプリ専用領域を削除する

なお、下のフロー図は、NFP タグのマウントまでが正常に完了していることを前提にしています。

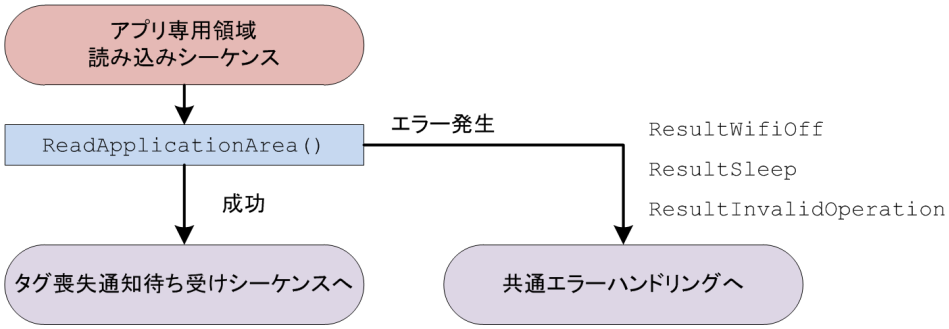
図 5-15. アプリ専用領域チェックシーケンスのフロー図



5.10.6. アプリ専用領域読み込みシーケンス

アプリ専用領域の読み込みを行うシーケンスです。

図 5-16. アプリ専用領域読み込みシーケンスのフロー図



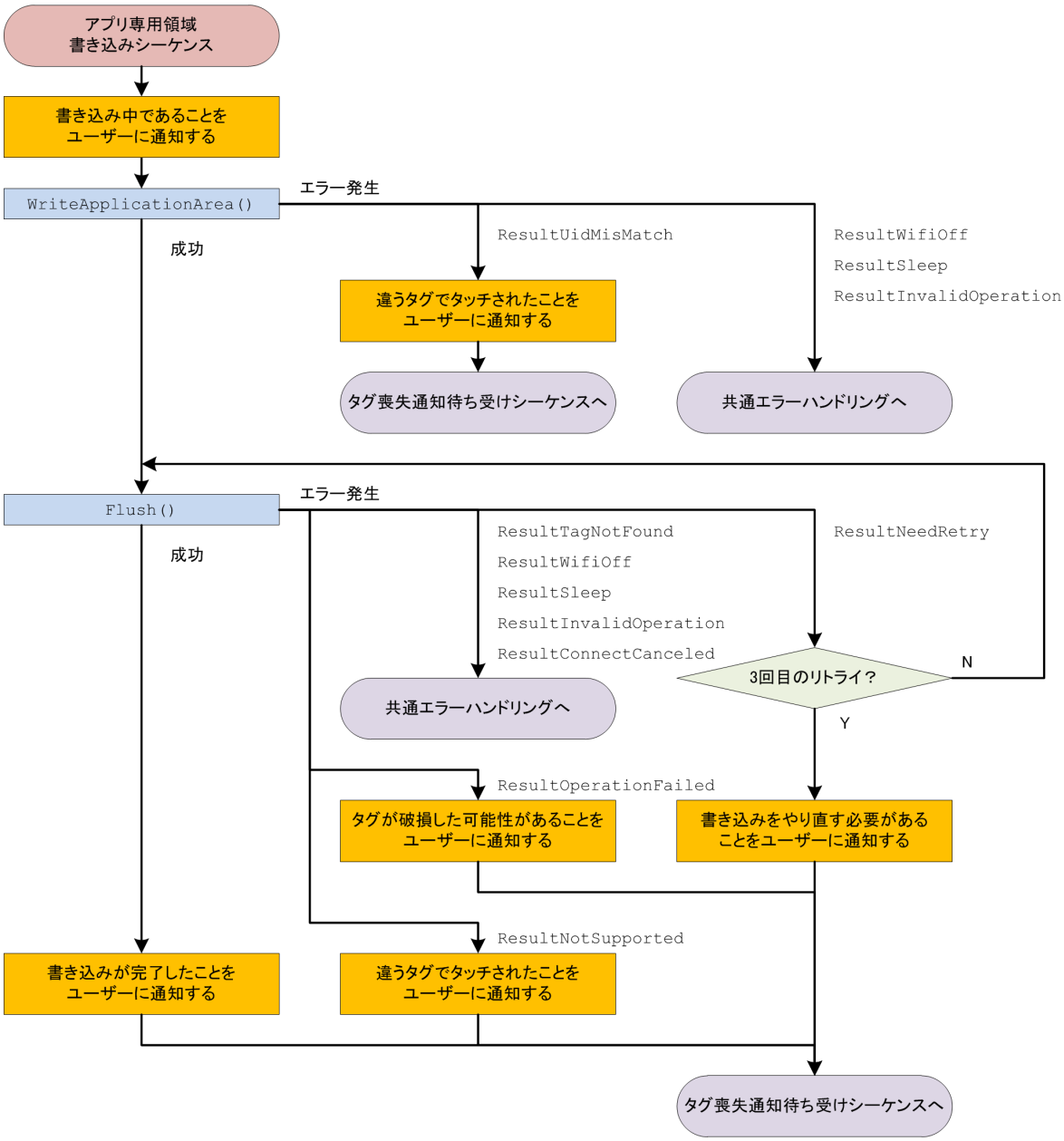
5.10.7. アプリ専用領域書き込みシーケンス

アプリ専用領域にデータを書き込むシーケンスです。

フロー図では、書き込みが完了したあとにタグが通信範囲外になることを待ち受けていますが、続けてタグの検知を再開しないのであれば `nn::nfp::StopDetection()` を呼び出し、次にタグ検知を再開するまでタグ検知を停止しておくことを推奨します。

処理でエラーが発生してもリトライが可能な場合の再試行数はフロー図で示した回数でなくてもかまいません。

図 5-17. アプリ専用領域書き込みシーケンスのフロー図



5.11. 3DS のアプリケーションで必要となる処理

3DS のアプリケーションでは、HOMEボタン、スリープ、電源ボタンといった 3DS の機能に対応した処理が必要になります。

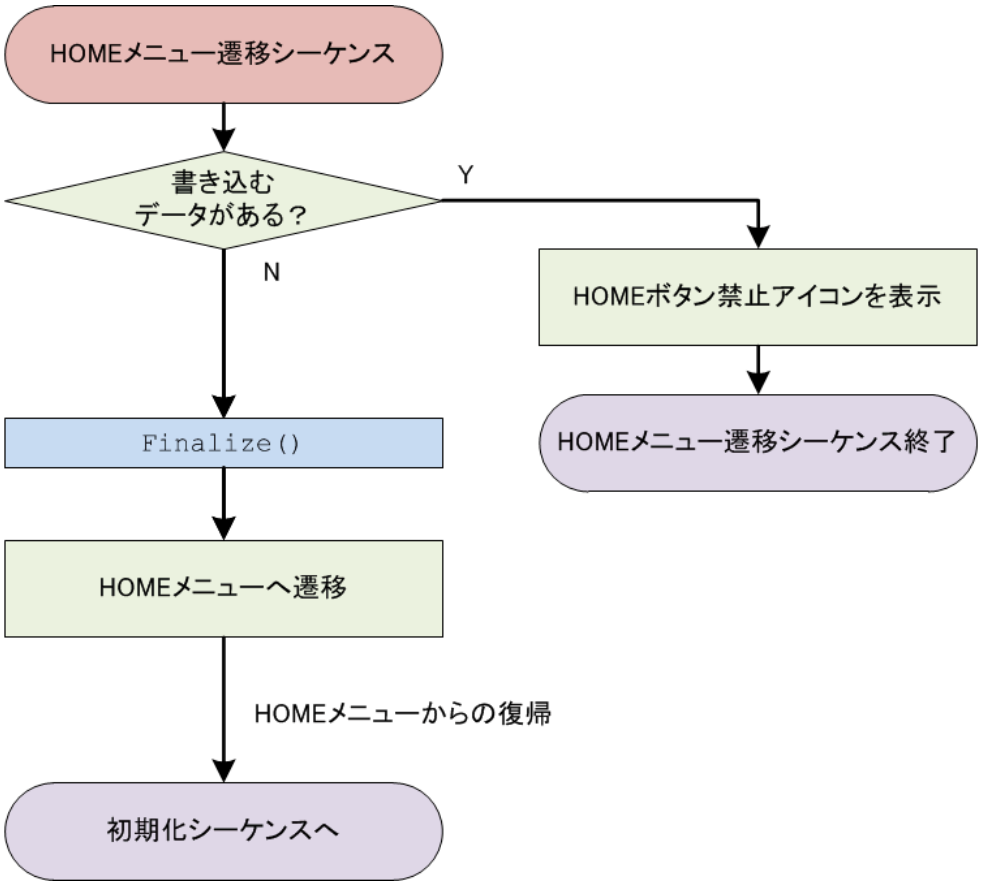
5.11.1. HOMEメニュー遷移シーケンス

HOMEボタンが押されると HOMEメニューへの遷移がシステムから要求され、アプリケーションは HOMEメニューへの遷移を行うかどうかを判断しなければなりません。

まだ書き込まれていないデータがあるなど、すぐに HOMEメニューへ遷移することができない場合は、HOMEボタン禁止アイコンを表示して HOMEメニューへの遷移を拒否することができます。

HOMEメニューから起動されるアプレットが NFC 機能を使用できるようにするために、NFP ライブラリを終了する必要があります。

図 5-18. HOMEメニュー遷移シーケンスのフロー図

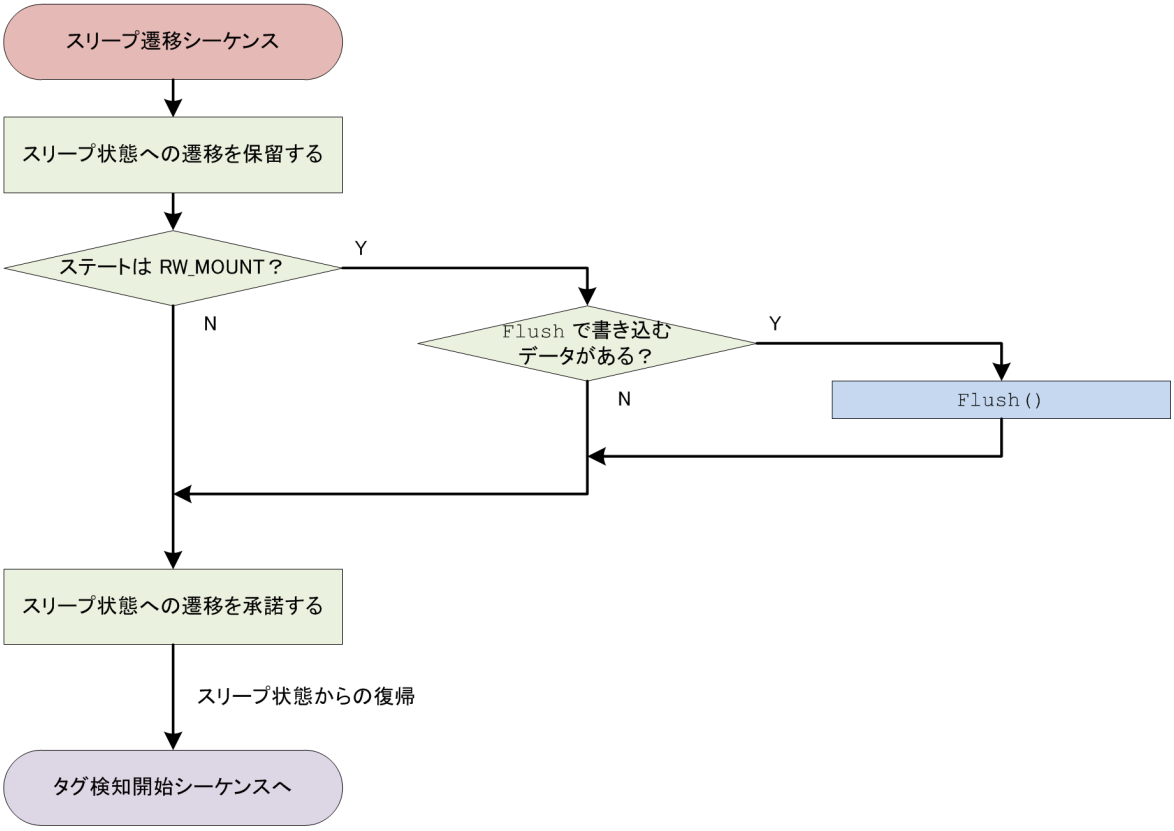


5.11.2. スリープ遷移シーケンス

蓋を閉じる(FTR はスリープスイッチをスライドする)ことで、アプリケーションにスリープ状態への遷移が要求され、アプリケーションはスリープ状態への遷移を承諾するかどうかを判断しなければなりません。

まだ書き込まれていないデータがあるなど、すぐにスリープ状態へ遷移することができない場合は、スリープ状態への遷移が可能になるまで要求への回答を保留して処理を続行することができます。アプリケーションがスリープ状態への遷移を承諾すると、システムによってタグの検知やマウントが自動的に解除されるため、スリープ状態から復帰した後は再びタグ検知開始シーケンスから始める必要があります。

図 5-19. スリープ遷移シーケンスのフロー図



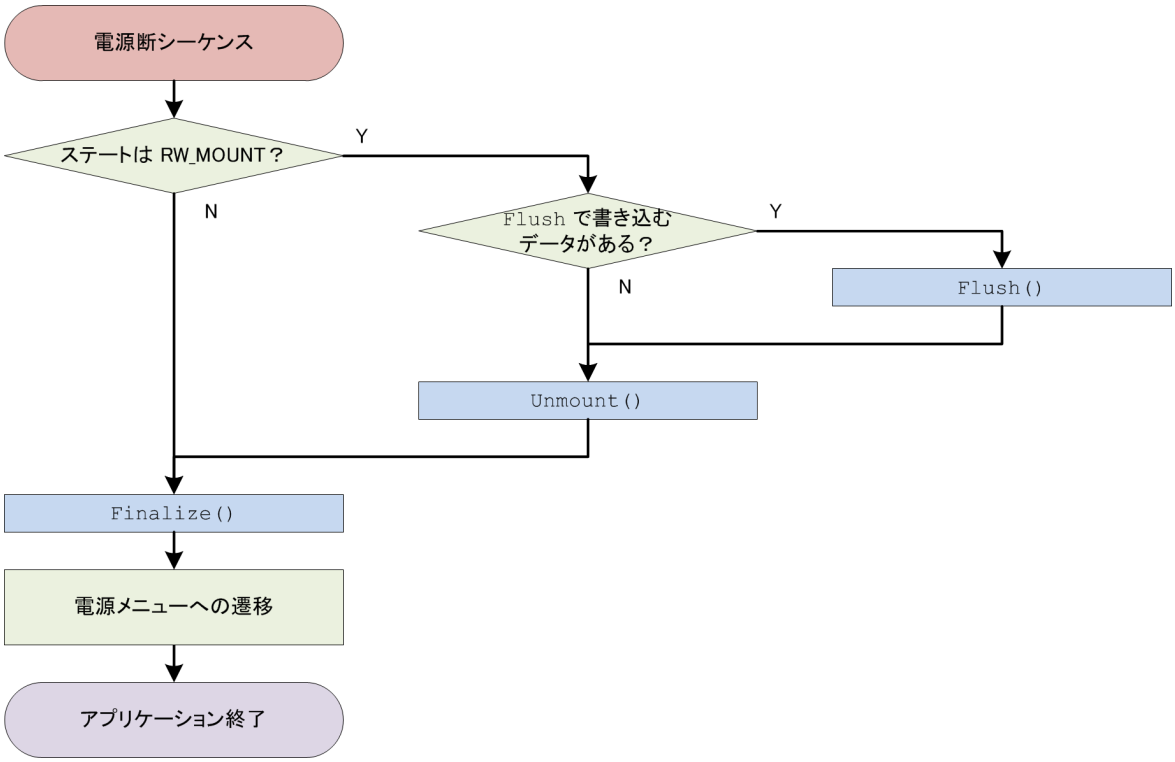
5.11.3. 電源断シーケンス

POWER ボタンが押されると電源メニューへの遷移がシステムから要求され、通常すぐにアプリケーションの終了処理を行わなければなりません。

まだ書き込まれていないデータがある場合は、アプリケーションに与えられた終了処理時間中に処理を完了できるならば、書き込み処理を続行することができます。

また、電源メニューを表示する前に NFP ライブラリを終了する必要があります。

図 5-20. 電源断シーケンスのフロー図



5.11.4. NFC リーダー/ライター接続シーケンス

CTR 向けの NFC リーダー/ライターを利用する場合は、アプリケーションから明示的に機器への接続を行わなければなりません。

スリープ状態や無線オフモードに遷移したことが原因で、 `nn::nfp::Connect()` でエラーが発生した場合は、戻り値に応じた処理を行ってください。
NFC リーダー/ライターに接続できなかった場合は、その原因を `nn::nfp::GetConnectResult()` で確認し、適切にハンドリングする必要があります。

`nn::nfp::ResultUpdateRequired` が発生していた場合は、NFC リーダー/ライターのファームウェアを更新する必要があるため、「amiibo設定」を起動する必要があります。

`nn::nfp::ResultIrFunctionError` が発生していた場合は、CTR の赤外線通信モジュールの故障の疑いがあります。このエラーが発生した場合は、CTR 本体の電源入れ直しを案内するようにし、何度も発生するようであれば、サービスセンターに問い合わせるように誘導してください。

`nn::nfp::ResultNfcTargetError` が発生していた場合は、NFC リーダー/ライターの故障の疑いがあります。このエラーが発生した場合は、NFC リーダー/ライターの電源入れ直しを案内するようにし、何度も発生するようであれば、サー

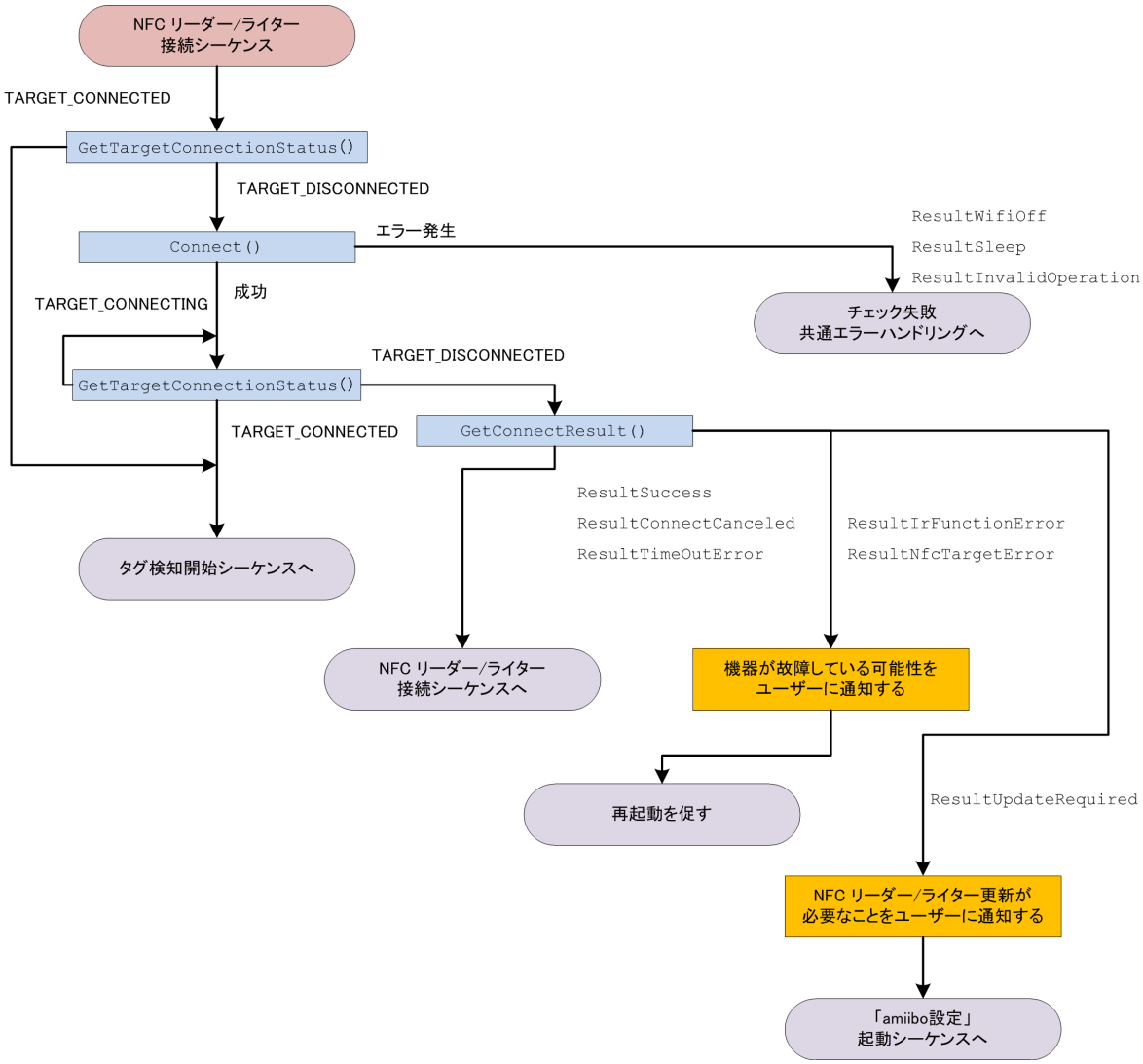
ビスセンターに問い合わせるように誘導してください。

`nn::nfp::ResultConnectCanceled` が発生していた場合は、接続がキャンセルされたか、内部エラーが発生しています。NFC リーダー/ライター接続をリトライしてください。
なお、このエラーはアプリケーション自身が `nn::nfp::Disconnect()` を呼び出して切断した場合にも発生します。

`nn::nfp::ResultTimeOutError` が発生していた場合は NFC リーダー/ライターからの返答がありません。NFC リーダー/ライターの電源を入れた直後や、置く場所がずれている可能性があります。NFC リーダー/ライター接続をリトライしてください。

接続に成功した後であっても、ユーザーが NFC リーダー/ライターを CTR から離す等して、通信出来ない状態にすると、1秒後に切断状態となります。そのため、定期的に `GetTargetConnectionStatus()` が `TARGET_DISCONNECTED` になっていないか、確認する必要があることに気をつけてください。

図 5-21. NFCリーダー/ライター接続シーケンスのフロー図



6. NFP ライブラリの利用方法

NFP ライブラリをアプリケーションで利用する際には、以下の点に注意して実装してください。

- NFP ライブラリは、SNAKE と CTR で挙動が変わる可能性があります。必ず SNAKE、CTR の両方で動作確認をお願いします。
- SNAKE 上で NFP ライブラリは、必ず本体内蔵 NFC を使用するモードで動作します。これはデバッガによる強制 CTR 互換モード時も同様です。NFC リーダー/ライターを使用する場合は CTR をご用意ください。
- NFP ライブラリはマルチスレッド対応されていません。同時に複数のスレッドから NFP ライブラリの関数を呼び出さないでください。
- 関数の実行に成功した場合は基本的に `nn::ResultSuccess` が返されますが、実行の成否判定は必ず `IsSuccess()` または `IsFailure()` で行ってください。
- `nn::nfp::ResultInvalidUsage` に属する返り値(`nn::nfp::ResultInvalidArgument`、`nn::nfp::ResultInvalidPointer`、`nn::nfp::ResultBufferIsSmall`)は実装エラーです。開発の段階でこれらのエラーが返されないように実装してください。
- NFP ライブラリで提供されている関数には、完了までに長時間要するものが多くあります。特にライブラリの初期化は最大で 5 秒程度、NFP タグへの書き込みは 1 ～ 2 秒程度の時間が必要です。そのため NFP ライブラリの関数は、メインループとは別のスレッドから呼び出すことを推奨します。
- タグを検知したあと、タグが通信可能範囲内から離れるなど、タグとの通信が行えない状態になるとタグ喪失通知がアプリケーションに通知されます。この時点で NFP ライブラリのステートは `RW_DEACTIVE` に遷移し、マウントされていたタグはアンマウントされます。
  この状態を把握せずに NFP ライブラリの関数を呼び出すと、ほとんどの関数は `nn::nfp::ResultInvalidOperation` を返します。アプリケーションはタグ喪失通知のハンドリングを必ず行い、検知していたタグにアクセスできる状態にあるかを NFP ライブラリの関数を呼び出す前に確認するようにしてください。
- NFP ライブラリの関数を実行している間に本体がスリープ状態や無線オフモードに遷移した場合、その多くは返り値として `nn::nfp::ResultSleep` または `nn::nfp::ResultWifiOff` を返します。これらの返り値が返されると、NFP ライブラリのステートは `INIT` に戻り、タグの検知は停止され、マウントされていたタグもアンマウントされた状態になっています。つまり、`nn::nfp::Unmount()` および `nn::nfp::StopDetection()` が呼ばれたのと同じ状態になります。
  スリープ状態から復帰したり無線オンモードに遷移したりしても NFP ライブラリは自動的にもとの状態に戻すことはありませんので、アプリケーションは `nn::nfp::StartDetection()` でタグの検知から再開する必要があります。
- HOMEメニューやスリープ状態に遷移する前に、アプリ専用領域への書き込みで `nn::nfp::Flush()` まで完了していない場合は、`nn::nfp::Flush()` による書き込みが完了するまで HOMEメニューやスリープ状態への遷移を遅らせても構いません。もしくは、NFP タグへの書き込みを行う間は HOMEボタン禁止アイコンを表示し、HOMEメニューへの遷移を行わないことで対処しても構いません。
- HOMEメニュー、及びアプレットに遷移する前に `nn::nfp::Finalize()` を呼び出して、NFP ライブラリのステートを NONE に遷移させてください。

NFP ライブラリの関数にはバックアップデータへのアクセスや NFP タグへのアクセスを行うものがあり、それらの関数を過度に呼び出すことはハードウェアや NFP タグの寿命を縮める原因となります。

そのため、ユーザーが意図的に操作を繰り返した場合を除いて、以下の関数を呼び出す頻度が過剰にならないように実装してください。

表 6-1. バックアップデータまたは NFP タグへのアクセスを行う関数

関数	バックアップデータからの読み込み	バックアップデータへの書き込み	NFP タグへの書き込み
nn::nfp::Initialize()	常時	バックアップ用のセーブデータが存在しない場合	なし
nn::nfp::Mount()	常時	初めてマウントするタグ、もしくは別の機器で書き換えられたタグをマウントした場合	なし
nn::nfp::Restore()	常時	なし	常時
nn::nfp::Flush()	なし	常時	常時
nn::nfp::CreateApplicationArea()	なし	常時	常時

6.1. 初期化

NFP ライブラリの初期化は nn::nfp::Initialize() を呼び出して行います。この関数の処理には通常数百ミリ秒程度、その本体で初めての実行では 5 秒程度の時間がかかるため、メインスレッド以外で呼び出すことを強く推奨します。また、アプリケーションの起動中ではなく、アプリケーションで画面更新が行えるようになってから呼び出すことを推奨します。

NFP ライブラリの関数のほとんどは初期化が完了する前に呼び出すと nn::nfp::ResultInvalidOperation を返します。初期化済みの状態で nn::nfp::Initialize() を呼び出した場合も nn::nfp::ResultInvalidOperation が返されますが、この場合については処理成功として扱うことができます。

NFP ライブラリの初期化に失敗したときに nn::nfp::ResultNeedRetry が返されることがあります。このエラーが返された場合は再度 nn::nfp::Initialize() を呼び出してください。なお、1 秒程度経過してから呼び出すことで初期化に成功する可能性が高くなります。

表 6-2. nn::nfp::Initialize() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	初期化に成功しました。
nn::nfp::ResultInvalidOperation	すでに初期化済みです。
nn::nfp::ResultNeedRetry	リトライしてください。
nn::nfp::ResultIrFunctionError	IrDA モジュールが故障している可能性があります。

6.1.1. タグ検知に関する通知を受け取るイベントの設定

タグ検知を開始する前に、タグ検知に関連する NFP ライブラリからの通知をアプリケーションで受け取るイベントを設定する必要があります。

NFP ライブラリからアプリケーションへは、以下のタイミングでタグ検知に関連する通知が行われます。

- NFC モジュールの通信可能範囲にタグを発見した(タグ発見通知)
- 発見していたタグが通信可能範囲から離れた(タグ喪失通知)
- タグを検知している状態で nn::nfp::StopDetection(), nn::nfp::Disconnect() または

nn::nfp::Finalize() を呼び出した(タグ喪失通知)

タグ発見通知に対する設定は nn::nfp::SetActivateEvent() で行い、タグ喪失通知に対する設定は nn::nfp::SetDeactivateEvent() で行います。

それぞれの関数に渡す引数は nn::os::Event クラスのインスタンスへのポインタです。NFP ライブラリがイベントの管理を行いますので、渡すインスタンスの初期化や解放はアプリケーションで行う必要はありません。

これらの関数はタグ検知を開始する前(ステートが INIT である間)に呼び出す必要があります。

表 6-3. nn::nfp::SetActivateEvent() および nn::nfp::SetDeactivateEvent() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	イベントの設定に成功しました。
nn::nfp::ResultInvalidPointer	引数で渡されたポインタが正しく指定されていません。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。

6.1.1.1. タグ発見通知に対する処理

NFP ライブラリがタグを発見したことは、nn::nfp::SetActivateEvent() で設定したイベントがシグナル状態になることでアプリケーションに通知されます。

タグが発見された段階で、以下の処理が可能になります。

- タグ情報の取得
- タグのマウント

6.1.1.2. タグ喪失通知に対する処理

通信可能範囲から離れるなどによって、NFP ライブラリが発見していたタグを見失ったことは、nn::nfp::SetDeactivateEvent() で設定したイベントがシグナル状態になることでアプリケーションに通知されます。この通知を受け取った時点で NFP ライブラリのステートが RW_DEACTIVE に遷移しているため、タグの検知を継続する場合は nn::nfp::StartDetection() を再度呼び出す必要があります。

タグを見失った原因は nn::nfp::GetConnectionStatus() で取得することができるタグの接続状態から判断することができます。タグの接続状態は nn::nfp::ConnectionStatus 構造体で定義されており、nn::nfp::GetConnectionStatus() には構造体へのポインタを渡す必要があります。

nn::nfp::GetConnectionStatus() の処理に成功すると、引数で渡した構造体に現在のタグの接続状態が設定されます。接続状態の詳細(切断理由)は nn::nfp::DeactivateReason 列挙子で定義され、構造体の deactivateReason メンバに設定されています。

表 6-4. 接続状態の詳細(切断理由)

定義	接続状態	説明
BEFORE_ACTIVATION	未発見	まだタグを発見できていません。
TAG_IS_ACTIVE	接続中	タグへのアクセスが可能です。
DETECT_ERROR	接続失敗	正常なタグだと認識できていません。タグのデータが破損している可能性があります。

CONNECTION_LOST	切断	タグが通信可能範囲から離れた、通信状況が悪化したなどの理由により接続が維持できなくなりました。
UNKNOWN_DEACTIVATE_REASON	切断	ライブラリが認知できない理由で接続が切断されました。

表 6-5. nn::nfp::GetConnectionStatus() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	接続状態の取得に成功しました。
nn::nfp::ResultInvalidPointer	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。

6.2. タグ検知の開始

タグ検知の開始は nn::nfp::StartDetection() で行います。

タグ検知を行っている間、NFP ライブラリはタグを発見したかどうかや発見していたタグと通信可能であるかどうかを定期的に判定します。タグ検知の状態が変化する要因が発生してから、タグ発見がアプリケーションに通知されるまでには、SNAKE で最大 500 ミリ秒、CTR で最大1秒程度のタイムラグがあります。タグ喪失がアプリケーションに通知されるまでには最大 2 秒程度のタイムラグがあります。そのため、タグがタッチされたり、タグが離されたりしたタイミングをシビアに判定するような実装を行わないでください。また、SNAKE では下画面液晶の中央にタグが置かれることを前提としています。具体的には、液晶の中心から前後左右方向に ± 10mm 、上下方向に + 5mm が正確にタグを認識できる範囲となります。CLOSER でも同様です。このため、液晶の中央以外にタグを置くように誘導する遊び方やユーザーインタフェースの設計は避けてください。

SNAKE ではタグの検知に使用する電磁波がノイズとなり、タッチパネルからの入力が不安定になってしまいます。この状態でタッチパネルの入力を使用するとアプリケー ションの誤動作に繋がるため、SNAKE ではタグの検知中にタッチパネルを利用できないように制限が課されています。具体的には、nn::nfp::StartDetection() を実行してから nn::nfp::StopDetection() や nn::nfp::Finalize() でタグの検知が停止されるまでの間、nn::hid::CTR::TouchPanelStatus の status が常に 0 (ペンアップ)に設定されます。

CTR では NFC リーダー/ライターでタグを検知するため、SNAKE と同じような制限は課されていません。ただし、CTR でタッチパネルを使用する場合には SNAKE 向けに異なる UI を用意するなどして、CTR と SNAKE の両方でアプリケーションが操作できることを確認してください。

NFP ライブラリではタグを発見した瞬間に消費電力や発熱が最大になります。タグを置いた状態で nn::nfp::StartDetection() 、nn::nfp::StopDetection() を繰り返すと、タグの発見が短い期間で連続して発生することになり、SNAKE では設計上許容される発熱量を超えてシステムが不安定になります。CTR ではこの問題は発生しませんが、電力を消費することには変わりありません。そのため、SNAKE 、CTR 共に、高頻度で nn::nfp::StartDetection() を実行できないように制限が課されています。具体的には、10 秒という時間枠で nn::nfp::StartDetection() を任意のタイミングで 10 回実行できますが、11 回以上は実行できないようになっています。これは、nn::nfp::StartDetection() の実行に際して、10 回前の nn::nfp::StartDetection() の実行から 10 秒経過するまで、その nn::nfp::StartDetection() の実行をブロックすることで実現しています。既に 10 秒経過している場合や、過去の実行回数が 10 回未満の場合は、ブロックされません。

表 6-6. nn::nfp::StartDetection() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	タグ検知を開始しました。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。

6.3. タグ情報の取得

タグ情報の取得は nn::nfp::GetTagInfo() の呼び出しで行います。タグ情報は nn::nfp::TagInfo 構造体で定義されており、nn::nfp::GetTagInfo() には構造体へのポインタを渡す必要があります。

表 6-7. nn::nfp::GetTagInfo() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	タグ情報の取得に成功しました。
nn::nfp::ResultInvalidPointer	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。

補足: NFP ライブラリでは、タグの UID が取得できるのはタグ情報からのみとなっています。NFP タグのアプリ専用領域へのアクセスを行うアプリケーションは、事前に nn::nfp::GetTagInfo() でタグ情報を取得しておくことを推奨します。

タグ情報は NFP ライブラリでアクセス可能な種類のタグであれば取得できる情報のみが記録されているため、発見されたタグが NFP タグであるかどうかまでは分かりません。

6.4. タグのマウント

NFP タグの情報にアクセスするには、まず nn::nfp::Mount() または nn::nfp::MountRom() を呼び出して、発見されたタグをマウントする必要があります。

ともに NFP タグでなければマウントが行われませんが、nn::nfp::MountRom() でマウントした場合はアクセス可能な情報に制限があり、タグのデータが壊れていてもマウントが可能な点が異なります。

以下のように、発見されたタグが NFP タグであるかどうかを返り値で判断することができます。

表 6-8. タグの状態と各関数の返り値

タグの状態	nn::nfp::Mount()	nn::nfp::MountRom()
NFP タグではない	nn::nfp::ResultNotSupported	
NFP タグだが未知のフォーマットバージョン	nn::nfp::ResultInvalidFormatVersion	
NFP タグであり、タグのデータが正常	nn::ResultSuccess	
NFP タグであり、タグのデータが壊れている (タグのバックアップデータが存在しない)	nn::nfp::ResultNeedFormat	nn::ResultSuccess
NFP タグであり、タグのデータが壊れている (タグのバックアップデータが存在する)	nn::nfp::ResultNeedRestore	nn::ResultSuccess

発見されたタグが NFP タグでない場合は nn::nfp::ResultNotSupported が返されます。

発見されたタグが NFP タグであっても、未知のフォーマットバージョンでありマウントできない場合には nn::nfp::ReultInvalidFormatVersion が返されます。

発見されたタグが NFP タグであり、タグのマウントに成功した場合は nn::ResultSuccess が返されます。

発見されたタグが NFP タグであり、タグのデータが壊れている場合は nn::nfp::ResultNeedRestore もしくは nn::nfp::ResultNeedFormat が返されます。前者の返り値に対しては nn::nfp::Restore() を呼び出してアプリケーションでタグのデータを修復することができます。後者の返り値に対してはアプリケーションでタグのデータを修復することができませんので、NFP タグの初期化を「amiibo設定」で行う必要があります。

返り値に nn::nfp::ResultTagNotFound が返された場合は NFP ライブラリのステートが RW_DEACTIVE に遷移しているため、タグの検知を継続する場合は nn::nfp::StartDetection() を再度呼び出す必要があります。

返り値に nn::nfp::ResultNeedRetry が返された場合は、再度 nn::nfp::Mount() を呼び出すことでマウントに成功する可能性があります。

> **補足:** タグのマウントには最大 500 ミリ秒程度必要です。
> リトライする場合は 100 ミリ秒程度の間隔をあけて行うことを推奨します。

表 6-9. nn::nfp::Mount() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	タグのマウントに成功しました。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultNotSupported	NFP タグではありません。
nn::nfp::ResultNeedRestore	タグのデータが壊れているため、nn::nfp::Restore() でタグのデータを修復してください。
nn::nfp::ResultNeedFormat	タグのデータが壊れているため、「amiibo設定」で初期化する必要があります。

nn::nfp::ResultTagNotFound	タグが見つかりませんでした。タグが通信可能範囲から離されたり、別のタグに変えられていたりすることが原因です。
nn::nfp::ResultNeedRetry	マウント中にエラーが発生しました。再度実行することで処理に成功する可能性があります。
nn::nfp::ResultInvalidFormatVersion	対応していないバージョンのタグです。対応していないタグであることをユーザーに通知してください。

表 6-10. nn::nfp::MountRom() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	タグのマウントに成功しました。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultNotSupported	NFP タグではありません。
nn::nfp::ResultTagNotFound	タグが見つかりませんでした。タグが通信可能範囲から離されたり、別のタグに変えられていたりすることが原因です。
nn::nfp::ResultInvalidFormatVersion	対応していないバージョンのタグです。対応していないタグであることをユーザーに通知してください。

6.4.1. タグの復旧

nn::nfp::Mount() が nn::nfp::ResultNeedRestore を返した場合、nn::nfp::Restore() を呼び出して、アプリケーションで NFP タグの復旧を行うことができます。

返り値に nn::nfp::ResultNeedRetry が返された場合は再度 nn::nfp::Restore() を呼び出すことでタグの復旧に成功する可能性があります。

> **補足:** タグの復旧には数秒程度かかる可能性があります。
> リトライする場合は 100 ミリ秒程度の間隔をあけて行うことを推奨します。

復旧されるデータは、タグのマウント時およびタグへの書き込みを行う直前に NFP ライブラリがバックアップしたデータです。そのため、NFP タグへの書き込みに失敗したあとに復旧した場合は、書き込み前のデータではなく、書き込もうとしたデータで復旧されることになります。

表 6-11. nn::nfp::Restore() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	タグの復旧に成功しました。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。

nn::nfp::ResultNotSupported	NFP タグではありません。
nn::nfp::ResultNotBroken	タグが壊れていないため、復旧する必要はありません。
nn::nfp::ResultTagNotFound	タグが見つかりませんでした。タグが通信可能範囲から離されたり、別のタグに変えられていたりすることが原因です。
nn::nfp::ResultOperationFailed	書き込み中にエラーが発生しました。タグの内容が正常に復旧できなかった可能性があります。
nn::nfp::ResultNeedRetry	書き込み処理の過程でエラーが発生しました。再実行してください。
nn::nfp::ResultConnectCanceled	内部エラーにより NFC リーダー/ライターとの接続が切断されました。
nn::nfp::ResultBackupError	バックアップデータへのアクセスに失敗しました。この NFP タグは復旧できません。

6.5. 「amiibo設定」の起動

「amiibo設定」の起動には以下の手順が必要です。

1. 「amiibo設定」の起動に必要な情報を設定する。
2. NFP ライブラリの終了処理を行う。
3. 「amiibo設定」を起動する。

最初に、「amiibo設定」の起動に必要な情報を nn::nfp::Parameter 構造体に設定します。

nn::nfp::InitializeParameter() で初期化してから、アプリケーションから「amiibo設定」を呼び出す際に必要な情報を input メンバ(nn::nfp::Input 構造体)に設定してください。

「amiibo設定」内でタグの検知を行う際にアプリケーションから指定されたタグであるかどうかを判定する必要があります。そのため、必ず tagInfo には nn::nfp::GetTagInfo() で事前に取得したタグ情報を設定してください。

そのほかの mode、isRegistered、registerInfo、commonInfo の各メンバには、ユースケースごとに以下の値を設定してください。

表 6-12. 「amiibo設定」を起動する際のユースケースごとの設定値

ユースケース	mode	isRegistered	registerInfo	commonInfo
nn::nfp::Mount() が nn::nfp::ResultNeedRestore を返した	AMIIBO_SETTINGS_RESTORE	false	NULL	NULL
nn::nfp::GetNfpRegisterInfo() が nn::nfp::ResultNeedRegister を返した	AMIIBO_SETTINGS_NICKNAME_OWNER	false	NULL	nn::nfp::GetNfpCommonInfo() で取得した情報
nn::nfp::GetNfpRegisterInfo() が nn::ResultSuccess を返したが、ニックネームが空文字列だった	AMIIBO_SETTINGS_NICKNAME_OWNER	true	nn::nfp::GetNfpRegisterInfo() で取得した情報	nn::nfp::GetNfpCommonInfo() で取得した情報
nn::nfp::OpenApplicationArea() が nn::nfp::ResultAccessIdMisMatch を返したので、アプリ専用領域を削除したい	AMIIBO_SETTINGS_ERASE_GAMEDATA	true	nn::nfp::GetNfpRegisterInfo() で取得した情報	nn::nfp::GetNfpCommonInfo() で取得した情報

mode メンバに設定する値は nn::nfp::AmiiboSettingsMode 列挙子で定義されており、「amiibo設定」を起動する際のモードを指定するためのものです。

表 6-13.「amiibo設定」の起動モード

定義	説明
AMIIBO_SETTINGS_RESTORE	タグを復旧します。
AMIIBO_SETTINGS_NICKNAME_OWNER	オーナー登録とニックネームの設定を行います。
AMIIBO_SETTINGS_ERASE_GAMEDATA	アプリ専用領域の削除を行います。

「amiibo設定」の呼び出しは nn::nfp::StartAmiiboSettings() で行います。この関数を呼び出す前に nn::nfp::Finalize() で NFP ライブラリの終了処理を行う必要があります。そのため、「amiibo設定」から戻ったあとに NFP ライブラリを利用する場合は NFP ライブラリの初期化から再開しなければなりません。

nn::nfp::StartAmiiboSettings() が false を返した場合は、「amiibo設定」の起動に失敗しています。エラーの原因としては、以下のものが考えられます。

- 事前に NFP ライブラリの終了処理(nn::nfp::Finalize() の呼び出し)を行っていない。
- 「amiibo設定」がインストールされていない(nn::nfp::IsAmiiboSettingsAvailable() が false を返す)。
- パラメータ設定が間違っている。
- アプレットの起動に失敗した。

なお、Parameter.output.result には「amiibo設定」で NFP タグへの変更が行われたかどうかが格納されます。このメンバが nn::nfp::AMIIBO_SETTINGS_RESULT_SUCCESS ならば「amiibo設定」で NFP タグへの変更が行われたことを示し、それ以外ならばユーザーが処理をキャンセルするなどして NFP タグへの変更が行われていないことを示しています。

ほかのライブラリアプレットと同様に、nn::nfp::StartAmiiboSettings() から制御が戻されたときには、「amiibo設定」の動作中に電源ボタンが押下されるなどでアプリケーションの終了要求が発生していないかを nn::applet::IsExpectedToProcessPowerButton() および nn::applet::IsExpectedToCloseApplication() でチェックしてください。

6.6. NFP タグへのアクセス

タグのマウントが成功すると、NFP タグの以下の情報にアクセスすることができるようになります。

- 共用領域
- アプリ専用領域

> **注意:** アプリ専用領域は領域を作成したアプリケーションと同じアクセスID を持つアプリケーションでなければアクセスすることができません。
>
> nn::nfp::MountRom() でマウントした場合は、共用領域の一部の情報にのみアクセスすることができます。

6.6.1. 共用領域へのアクセス

マウントされた NFP タグの情報をアプリケーションで使用する前に、nn::nfp::GetNfpCommonInfo() または nn::nfp::GetNfpRomInfo() で共用領域の情報を取得し、アプリケーションが対応している NFP タグかどうかを判断してください。

> **注意:** nn::nfp::MountRom() でマウントしていた場合、nn::nfp::GetNfpCommonInfo() は nn::nfp::ResultInvalidOperation を返します。

nn::nfp::GetNfpCommonInfo() で取得する共用領域の情報は nn::nfp::CommonInfo 構造体で定義されており、関数には構造体へのポインタを渡す必要があります。

nn::nfp::GetNfpRomInfo() で取得する共用領域の情報は nn::nfp::RomInfo 構造体で定義されており、関数には構造体へのポインタを渡す必要があります。

アプリケーションが対応している NFP タグであるかどうかはキャラクターID(characterId)のみで判断してください。なお、対応するキャラクターID のリストはアプリケーションで用意する必要があります。

> **注意:** キャラクターID 以外の情報をアプリケーションで使用することは、現在禁止されています。

表 6-14. nn::nfp::GetNfpCommonInfo() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	情報の取得に成功しました。
nn::nfp::ResultInvalidPointer	引数で渡されたポインタが正しく指定されていません。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。

表 6-15. nn::nfp::GetNfpRomInfo() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	情報の取得に成功しました。
nn::nfp::ResultInvalidPointer	引数で渡されたポインタが正しく指定されていません。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。

共用領域には nn::nfp:GetNfpCommonInfo() や nn::nfp::GetNfpRomInfo() で取得する情報のほかに、nn::nfp::GetNfpRegisterInfo() で取得する登録情報があります。

> **注意:** nn::nfp::MountRom() でマウントしていた場合、nn::nfp::GetNfpRegisterInfo() は nn::nfp::ResultInvalidOperation を返します。

登録情報は nn::nfp::RegisterInfo 構造体で定義されており、nn::nfp::GetNfpRegisterInfo() には構造体へのポインタを渡す必要があります。

返り値に nn::nfp::ResultNeedRegister が返された場合、マウントされた NFP タグは「amiibo設定」によるオーナー登録が行われていません。登録情報を必須とするアプリケーションは「amiibo設定」を起動してください。

> **補足:** 「amiibo設定」の起動方法については「6.5. 「amiibo設定」の起動」を参照してください。

取得した登録情報にはオーナーとして登録された Mii のデータ(miiData)、amiibo のニックネーム(nickName)、ニックネームの表示に必要なフォントのリージョン(fontRegion)が記録されています。

Mii のデータをアプリケーションで利用するには、似顔絵ライブラリが必要です。アプリケーションで Mii を表示する方法については似顔絵ライブラリのドキュメントを参照してください。

ニックネームは UTF-16 BE で格納されていますので、必要に応じてエンディアンの変換などを行ってください。なお、終端文字は NULL(0x0000)です。ニックネームをアプリケーションで表示する際には、フォントリージョンで指定された内蔵フォントで表示するか、アプリケーションで使用しているフォントにない文字を代替文字で表示するなどの対応を行い、画面表示が崩れたり、アプリケーションの進行に影響が出たりしないようにしてください。

オーナー登録がされていれば、Mii のデータは必ず設定されていますが、ニックネームは設定されていない(空文字列である)場合があります。ニックネームの登録を必須とするアプリケーションは nn::nfp::ResultNeedRegister と同じ手順で「amiibo設定」を起動してください。

> **補足:** ニックネームを表示するアプリケーションは、amiibo のニックネームとオーナーの Mii のニックネームの扱いに注意してください。特に、ニックネームがオーナーの Mii の名前であるかのような表示にならないようにしてください。また、「(オーナーの名前)のamiibo」や「(amiibo のキャラクター名)」、「amiibo」、「アミーボ」のように、ニックネームが設定されていない場合でも代替となる文字列が表示されるようにしてください。

> **注意:** 国情報(country)をアプリケーションで使用することは、現在禁止されています。

表 6-16. nn::nfp::GetNfpRegisterInfo() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	情報の取得に成功しました。
nn::nfp::ResultInvalidPointer	引数で渡されたポインタが正しく指定されていません。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultNeedRegister	「amiibo設定」によるオーナー登録が行われていません。

6.6.2. アプリ専用領域へのアクセス

アプリ専用領域はアクセスID によってアクセス権限の制御が行われています。そのため、アプリ専用領域へのアクセスを開始するには、最初に nn::nfp::OpenApplicationArea() を呼び出して、アプリケーションにアクセス権限があるかを確認する必要があります。

返り値に nn::nfp::ResultAccessIdMisMatch が返された場合、アプリ専用領域は違うアクセスID によって作成されたものですので、アプリケーションにはアクセス権限がありません。この場合、読み込みおよび書き込みの両方が制限されます。アプリケーションからアプリ専用領域を削除する方法はなく、このアプリケーション用にアプリ専用領域を利用したい場合は「amiibo設定」で削除する必要があります。

返り値に nn::nfp::ResultNeedCreate が返された場合、マウントされた NFP タグにはアプリ専用領域がない状態ですので、アプリケーションでアプリ専用領域を作成してください。

表 6-17. nn::nfp::OpenApplicationArea() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。アプリ専用領域へのアクセスが可能です。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultAccessIdMisMatch	ほかのアクセスID で作成されたアプリ専用領域のため、アクセスすることができません。
nn::nfp::ResultNeedCreate	アプリ専用領域が作成されていません。

6.6.2.1. アプリ専用領域の作成

アプリ専用領域の作成は nn::nfp::CreateApplicationArea() を呼び出して行います。

この関数に渡す nn::nfp::ApplicationAreaCreateInfo 構造体は、
nn::nfp::InitializeCreateInfo() で初期化してから構造体の各メンバに値を設定してください。

アプリ専用領域に書き込むデータを CAFE でも読み込む場合、CAFE と CTR/SNAKE ではエンディアンが違うため、どちらかに統一することを推奨します。

> **補足:** アプリ専用領域の作成には数秒程度かかる可能性があります。
>
> リトライする場合は 100 ミリ秒程度の間隔をあけて行うことを推奨します。
>
> 関数内で nn::nfp::Flush() と同じ動作を行っているため、
> nn::nfp::CreateApplication() に続けて nn::nfp::Flush() を呼び出す必要はありません。

表 6-18. nn::nfp::CreateApplicationArea() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。アプリ専用領域は作成されました。

nn::nfp::ResultInvalidArgument	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidPointer	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultNotSupported	対応していないタグです。別のタグに変えられている可能性があります。
nn::nfp::ResultAlreadyCreated	すでにアプリ専用領域が作成されています。
nn::nfp::ResultTagNotFound	タグが見つかりませんでした。タグが通信可能範囲から離されたり、別のタグに変えられていたりすることが原因です。
nn::nfp::ResultNeedRetry	書き込み中にエラーが発生しました。再度実行することで処理に成功する可能性があります。
nn::nfp::ResultOperationFailed	書き込み中にエラーが発生しました。タグの内容が破損した可能性があります。
nn::nfp::ResultConnectCanceled	内部エラーにより NFC リーダー/ライターとの接続が切断されました。

以下のサンプルコードではリトライ回数を 3 回までとしています。

コード 6-1. アプリ専用領域の作成（サンプルコード）

```
#define MY_ACCESS_ID 0x12345678
#define MY_APPLICATION_AREA_SIZE 128
nn::Result result;
bit8 writeData[MY_APPLICATION_AREA_SIZE];

// 最初に設定情報を初期化
nn::nfp::ApplicationAreaCreateInfo createInfo;
result = nn::nfp::InitializeCreateInfo(&createInfo);
if (result.IsFailure())
{
    // エラー
    return;
}

// 初期化データを用意
{
    memset(writeData, NULL, sizeof(writeData));
    // ビッグエンディアンでデータを書き込みます。
    writeData[0] = 'I';
    writeData[1] = 'N';
    writeData[2] = 'I';
    writeData[3] = 'T';
    writeData[4] = 0x00;
    writeData[5] = 0x00;
    writeData[6] = 0x00;
    writeData[7] = 0x01;
```

```
}
// 初期化に必要なデータを設定
createInfo.accessId = MY_ACCESS_ID;
createInfo.pInitialData = writeData;
createInfo.initialDataSize = sizeof(writeData);

// アプリ専用領域の作成
int retryCount = 0;
while (true)
{
    result = nn::nfp::CreateApplicationArea(createInfo);
    if (result.IsSuccess()) break;
    if (result.IsFailure())
    {
        if (nn::nfp::ResultNeedRetry::Includes(result))
        {
            // リトライは3回まで
            retryCount++;
            if (retryCount < 3)
            {
                // 0.1秒まってからリトライ
                nn::os::Thread::Sleep(100);
                continue;
            }
            // 処理失敗。もう一度タグの検知からやり直すなどの対処が必要
        }
        else if (nn::nfp::ResultAlreadyCreated::Includes(result))
        {
            // すでにアプリ専用領域が作成済み
            // データの整合性を確認して問題がなければ成功
        }
        else if (nn::nfp::ResultNotSupported::Includes(result))
        {
            // タグが差し替えられた可能性あり
        }
        else if (nn::nfp::ResultOperationFailed::Includes(result))
        {
            // タグが破損した可能性あり
        }
        else if (nn::nfp::ResultTagNotFound::Includes(result))
        {
            // タグが離された可能性あり
        }
        else if (nn::nfp::ResultInvalidOperation::Includes(result))
        {
            // 呼び出せないステートに遷移している
        }
        else if (nn::nfp::ResultWifiOff::Includes(result))
        {
            // 無線オフモードのため失敗
        }
        else if (nn::nfp::ResultSleep::Includes(result))
        {
```

```
            // スリープモードのため失敗
        }
        else if (nn::nfp::ResultConnectCanceled::Includes(result))
        {
            // NFC リーダー/ライターとの接続が切断したため失敗
        }
        else
        {
            // そのほかのエラー
        }
        return;
    }
}
// nn::nfp::Flush() を呼び出す必要はありません
```

6.6.2.2. アプリ専用領域の読み込み

アプリ専用領域に書き込まれているデータを読み込むには nn::nfp::ReadApplicationArea() を呼び出します。

アプリ専用領域の先頭から指定されたバイト数の内容をバッファに読み込むことができます。アプリ専用領域から読み込み可能なデータの最大サイズは nn::nfp::GetCommonInfo() で取得した CommonInfo 構造体の applicationAreaSize メンバから取得することができます。

表 6-19. nn::nfp::ReadApplicationArea() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。
nn::nfp::ResultInvalidArgument	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidPointer	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。

6.6.2.3. アプリ専用領域への書き込み

アプリ専用領域への書き込みは nn::nfp::WriteApplicationArea() と nn::nfp::Flush() を呼び出して行います。

nn::nfp::WriteApplicationArea() の実行には、書き込み対象の NFP タグの UID が必要です。事前に nn::nfp::GetTagInfo() で取得した TagInfo 構造体の tagId メンバを引数に渡してください。アプリ専用領域に書き込むデータは、NFP ライブラリがタグに書き込む情報に合わせて、ビッグエンディアンにすることを推奨します。

nn::nfp::WriteApplicationArea() の実行が完了した時点では、まだ NFP タグのアプリ専用領域への書き込みは行われず、NFP ライブラリ内でキャッシュされている状態になっています。NFP タグのアプリ専用領域への書き込みは nn::nfp::Flush() で行われます。

補足： NFP タグのアプリ専用領域への書き込みには数秒程度かかる可能性があります。
リトライする場合は 100 ミリ秒程度の間隔をあけて行うことを推奨します。

表 6-20. nn::nfp::WriteApplicationArea() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。
nn::nfp::ResultInvalidArgument	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidPointer	引数で渡された情報に不備があり、処理に失敗しました。このエラーは実装エラーですので、開発の段階で発生しないように対処してください。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultUidMisMatch	指定された UID とタグの UID が異なります。

表 6-21. nn::nfp::Flush() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultNotSupported	対応していないタグです。別のタグに変えられている可能性があります。
nn::nfp::ResultTagNotFound	タグが見つかりませんでした。タグが通信可能範囲から離されたり、別のタグに変えられていたりすることが原因です。
nn::nfp::ResultOperationFailed	書き込み中にエラーが発生しました。タグの内容が破損した可能性があります。
nn::nfp::ResultNeedRetry	書き込み中にエラーが発生しました。再度実行することで処理に成功する可能性があります。
nn::nfp::ResultConnectCanceled	内部エラーにより NFC リーダー/ライターとの接続が切断されました。

6.7. タグのマウント解除

タグのマウント解除は nn::nfp::Unmount() で行います。

表 6-22. nn::nfp::Unmount() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。

6.8. タグ検知の終了

タグ検知の終了は nn::nfp::StopDetection() で行います。

タグがマウントされている状態で呼び出された場合、NFP ライブラリ内でマウントの解除が行われます。

なお、タグが検知された状態で nn::nfp::StopDetection() を呼び出した場合は、タグ喪失通知がアプリケーションに通知されます。

表 6-23. nn::nfp::StopDetection() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。
nn::nfp::ResultInvalidOperation	ライブラリのステートがこの関数を呼び出すのにふさわしくありません。
nn::nfp::ResultSleep	スリープ状態に遷移しているため、処理に失敗しました。
nn::nfp::ResultWifiOff	無線オフモードのため、処理に失敗しました。
nn::nfp::ResultConnectCanceled	内部エラーにより NFC リーダー/ライターとの接続が切断されました。
nn::nfp::ResultIrFunctionError	IrDA モジュールが故障している可能性があります。

6.9. 終了

NFP ライブラリの終了処理は nn::nfp::Finalize() を呼び出して行います。関数内でタグのマウント解除やタグ検知の終了が行われ、タグを検知していた場合はタグ喪失通知の通知も行われます。

表 6-24. nn::nfp::Finalize() が返す可能性のある返り値

返り値	説明
nn::ResultSuccess	処理に成功しました。
nn::nfp::ResultInvalidOperation	すでに終了済みです。

6.10. 3DS のアプリケーションで利用する場合

3DS のアプリケーションで NFP ライブラリを利用する場合、HOMEボタン、スリープ、電源ボタンといった 3DS の機能に対応した実装が必要になります。

CTR 向けの NFC リーダー/ライターは、CTR 本体の赤外線通信を利用しています。そのため、機器の接続および切断、機器からの通知のハンドリングを行う必要があります。

SNAKE, CTR 共に影響がある問題として、システムに負荷がかかっている場合、NFP ライブラリの動作に問題が発生することがあります。

6.10.1. HOMEボタンへの対処

HOME ボタンが押されたことを検知した場合は、HOME メニューへ遷移する前に `nn::nfp::Finalize()` を呼び出して、NFP ライブラリのステートを NONE に遷移させてください。アプレットを起動する前にも同じ操作が必要です。この操作を忘れた場合、Release ビルドではシステムが自動的に NFP ライブラリを NONE 状態に遷移させますが、Debug ビルドや Development ビルドではアサートが発生してプログラムの動作が停止します。

アプリ専用領域にデータを書き込んでいる最中など、NFP タグのデータを保護する必要がある場合は HOME ボタン禁止アイコンを表示して HOME メニューへの遷移を行わなくてもかまいません。通常、データの書き込みは 1 ～ 2 秒で終了します。

コード 6-2. HOMEボタンへの対処のサンプルコード

```
// HOME メニューへの遷移をチェック
if (nn::applet::IsExpectedToProcessHomeButton())
{
    // NFP タグへの書き込みを行っているならば HOME メニューに遷移しない
    if (_isNfpTagWritingInProgress())
    {
        // HOME ボタン禁止アイコンを表示する
        _displyHomeNixSign();
        nn::applet::ClearHomeButtonState();
    }
    else
    {
        // NFP タグへのアクセスを停止する
        // HOME メニューに遷移する前に nn::nfp::Finalize() で
        // ステートを NONE に遷移させる必要があります
        _stopNfpTagAccess();
        _finalizeNfpTagAccess();
        // HOME メニュー遷移前の処理
        _prepareHomeMenu();
        // HOME メニューに遷移する
        nn::applet::ProcessHomeButtonAndWait();
        if (nn::applet::IsExpectedToCloseApplication())
        {
            // アプリケーションを終了させる
            _closeApplication();
        }
        // HOME メニューからの復帰処理
        _returnFromHomeMenu();
        // すぐにタグの検知を再開するならばここで行う(nn::nfp::StartDetection()を呼び出す)
```

```
        if (_needRestartNfpTagAccess()) _startNfpTagAccess();
    }
}
```

6.10.2. スリープ要求への対処

NFP ライブラリはスリープ状態に遷移したときにステートを INIT にまで自動的に遷移させますが、アプリ専用領域にデータを書き込んでいる最中など、NFP タグのデータを保護する必要がある場合はアプリケーションでハンドリングしてスリープの承諾を保留することを推奨します。

コード 6-3. スリープ要求への対処のサンプルコード

```
// スリープ要求への対処
// スリープ問い合わせコールバック内で返答を保留していることを前提とする
if (nn::applet::IsExpectedToReplySleepQuery())
{
    // NFPタグへの書き込みを行っているかをチェック
    if (_isNfpTagWritingInProgress())
    {
        // NFPタグへの書き込みを行っているならばまだスリープしない
    }
    else
    {
        // スリープ前の処理
        _prepareSleep();
        // スリープへの遷移を承諾する
        nn::applet::ReplySleepQuery(nn::applet::REPLY_ACCEPT);
        // スリープ復帰コールバックなどでイベントをシグナル状態にするのを待つ
        _waitAwakeEvent();
        // スリープからの復帰処理
        _returnFromSleep();
        // すぐにタグの検知を再開するならばここで行う(nn::nfp::StartDetection()を呼び出す)
        if (_needRestartNfpTagAccess()) _startNfpTagAccess();
    }
}
```

6.10.3. 電源ボタンへの対処

電源ボタンが押された場合、アプリケーションは速やかに電源メニューを表示しなければなりません。電源メニューへ遷移する前に `nn::nfp::Finalize()` を呼び出して、NFP ライブラリのステートを NONE に遷移させてください。この操作を忘れた場合、Release ビルドではシステムが自動的に NFP ライブラリを NONE 状態に遷移させますが、Debug ビルドや Development ビルドではアサートが発生してプログラムの動作が停止します。

ただし、アプリ専用領域にデータを書き込んでいる最中など、NFP タグのデータを保護する必要がある場合は書き込みが完了するのを待ってから電源メニューを表示することができます。通常、データの書き込みは 1 ～ 2 秒で終了します。

コード 6-4. 電源ボタンへの対処のサンプルコード

```
// 電源ボタンへの対処
if (nn::applet::IsExpectedToProcessPowerButton())
{
    // NFP タグへの書き込みを行っているかをチェック
    if (_isNfpTagWritingInProgress())
```

```
    {
        // NFP タグへの書き込みが終了するまでは電源ボタンをハンドリングしない
    }
    else
    {
        // NFP タグへのアクセスを停止する
        // 電源メニューに遷移する前に nn::nfp::Finalize() で
        // ステートを NONE に遷移させる必要があります
        _stopNfpTagAccess();
        _finalizeNfpTagAccess();
        // 電源メニューへの遷移前に必要な処理を行う
        _preparePowerButtonMenu();
        // 電源メニューを表示する
        nn::applet::ProcessPowerButtonAndWait();
        if (nn::applet::IsExpectedCloseApplication())
        {
            // アプリケーションの終了処理
            _closeApplication();
        }
        else
        {
            // 電源メニューからの復帰
            _returnFromPowerButtonMenu();
        }
    }
}
```

6.10.4. CTR 対応に必要な実装

CTR 向けの NFC リーダー/ライターを利用する場合は、アプリケーションから明示的に機器への接続を行わなければなりません。また、タグの発見と喪失の通知のほかに、機器の接続と切断の状態をハンドリングする必要があります。

NFC リーダー/ライターの利用に関連する関数は以下の 4 つです。

- `nn::nfp::Connect()`
- `nn::nfp::GetTargetConnectionStatus()`
- `nn::nfp::GetConnectResult()`
- `nn::nfp::Disconnect()`

`nn::nfp::Connect()` は NFP ライブラリに対して機器への接続を要求します。この関数は非同期関数となっており、接続の成功、失敗は `nn::nfp::GetTargetConnectionStatus()` で取得した値が `nn::nfp::TARGET_CONNECTED` もしくは `nn::nfp::TARGET_DISCONNECTED` になっているかで判断します。

NFC リーダー/ライターと CTR が正しく向き合っている場合、接続が完了するまでには数百ミリ秒程度の時間がかかります。ただし、NFC リーダー/ライターの電源を入れた直後に接続を試みると、NFC リーダー/ライダー内部の初期化処理のために、接続が完了するまで 1 秒程度かかることがあります。NFC リーダー/ライターと CTR が向き合っていない場合は、タイムアウトまで接続を試みるため、1 秒後に `nn::nfp::TARGET_DISCONNECTED` 状態になります。

なお、SNAKE で `nn::nfp::GetTargetConnectionStatus()` を呼び出した場合は何も処理をせず、即座に `nn::nfp::TARGET_CONNECTED` が取得されます。

表 6-25. nn::nfp::GetTargetConnectionStatus() で取得できる値

取得した値	NFC リーダー/ライター との接続状態
nn::nfp::TARGET_DISCONNECTED	切断
nn::nfp::TARGET_CONNECTING	接続中
nn::nfp::TARGET_CONNECTED	接続

補足: ユーザーが CTR を持って遊んでいる場合等、NFC リーダー/ライターとの向きが合わなくなると、比較的短時間で切断されることになります。
ユーザーに切断を意識させたくない場合、アプリケーション内部で常に接続状態を維持するように、nn::nfp::TARGET_DISCONNECTED 状態であれば nn::nfp::Connect() を自動で呼び出すような作りにすることも可能です。

nn::nfp::GetConnectResult() は、接続が失敗した場合に、失敗原因を取得するために呼び出します。呼び出しは nn::nfp::GetTargetConnectionStatus() で取得した値が nn::nfp::TARGET_DISCONNECTED になった後に実行してください。

なお、SNAKE で nn::nfp::GetConnectResult() を呼び出した場合は何も処理をせず、即座に nn::ResultSuccess が取得されます。

表 6-26. nn::nfp::GetConnectResult() で取得できる失敗要因

取得した失敗要因	説明
nn::ResultSuccess	接続に成功、もしくは Connect() の呼び出しに関係のない切断が発生しました。
nn::nfp::ResultConnectCanceled	接続処理がキャンセル、もしくは内部エラーにより失敗しました。
nn::nfp::ResultTimeOutError	接続が成功せずタイムアウトしました。
nn::nfp::ResultUpdateRequired	NFC リーダー/ライターのアップデートが必要です。
nn::nfp::ResultIrFunctionError	IrDA モジュールが故障している可能性があります。
nn::nfp::ResultNfcTargetError	NFC リーダー/ライターが故障している可能性があります。

補足: NFC リーダー/ライターと接続が失敗した場合のエラーハンドリングについては 5.11.4. NFC リーダー/ライター接続シーケンス を参照してください。

nn::nfp::Disconnect() はアプリケーションから明示的に機器への接続を切断する場合に呼び出します。処理の完了までには数十ミリ秒程度の時間がかかります。nn::nfp::StartDetection() でタグの検知中であれば、内部で nn::nfp::StopDetection() を行って切断と同時にタグ検知を停止するので、処理の完了までの時間が合計 100 ミリ秒程度かかるようになります。

NFC リーダー/ライターから切断された状態で呼び出すと nn::nfp::ResultInvalidOperation を返しますので、nn::nfp::GetTargetConnectionStatus() で、NFC リーダー/ライターとの接続状況を正確に把握しておくことを推奨します。

なお、SNAKE で nn::nfp::Disconnect() を呼び出した場合は、内部で nn::nfp::StopDetection() のみを

行い、タグの検知を停止します。

> **補足:** NFC リーダー/ライターの電池残量チェックは、nn::nfp::Connect() で CTR と接続した直後に、一度だけ行いますが、以降は接続が切断されるまで行いません。
> その為、接続状態を保持したまま数分間連続でタグの検出、書き込みを続けると、電池残量警告である赤色LED点灯無しに、いきなり NFC リーダー/ライターの電源断が発生する可能性があります。
>
> 連続でタグの検出、書き込みをするような使い方をする場合、定期的(1分毎等)に nn::nfp::Disconnect, nn::nfp::Connect を繰り返して使用することを推奨します。
>
> また、使用している電池の種類によっては、上記電源断の発生後に電源を入れなおすと、LEDが青色に点灯することがありますが、こちらは仕様となります。
> そのまま NFC リーダー/ライターを使用した場合、再度電源断が発生する前に LED が赤色になります。

以下はタグ検知の開始と停止のサンプルコードです。

コード 6-5. タグ検知開始および停止のサンプルコード

```
bool _willTerminate = false;
bool _connected = false;
nn::Result _lastResult = nn::ResultSuccess();
nn::nfp::TargetConnectionStatus _targetConnectionStatus;

// タグ発見通知イベント、nn::nfp::SetActivateEvent() で設定
nn::os::Event _eventActivate;

// タグ喪失通知イベント、nn::nfp::SetDeactivateEvent() で設定
nn::os::Event _eventDeactivate;

// タグ検知の開始
nn::Result _startDetection()
{
    // CTR独自の実装
    if (!nn::os::IsRunOnSnake())
    {
        // 外付けNFCの接続
        if (!_connected)
        {
            _lastResult = nn::nfp::Connect();
            if (_lastResult.IsSuccess())
            {
                // 接続処理完了を待ち受ける
                while (1)
                {

nn::os::Thread::Sleep(nn::fnd::TimeSpan::FromMilliSeconds(500));
                    nn::nfp::GetTargetConnectionStatus(&_targetConnectionStatus);
                    if (_targetConnectionStatus == nn::nfp::TARGET_CONNECTED)
                    {
                        // 接続に成功
                        _connected = true;
                        break;
                    }
```

```
                    else if (_targetConnectionStatus ==
nn::nfp::TARGET_DISCONNECTED)
                    {
                        // 接続に失敗、もしくは接続タイムアウト
                        _connected = false;
                        nn::nfp::GetConnectResult(&_lastResult);
                        return _lastResult;
                    }
                }
            }
            else
            {
                if (nn::nfp::ResultInvalidOperation::Includes(_lastResult))
                {
                    // ステートがおかしい
                }
                else
                {
                    // そのほかのエラーはPANICで停止させる
                    NN_PANIC_WITH_RESULT(_lastResult);
                }
                return _lastResult;
            }
        }
    }

    // 検知開始
    _lastResult = nn::nfp::StartDetection();
    if (_lastResult.IsSuccess())
    {
        // 処理成功
        // タグでタッチするように促す
    }
    else if (nn::nfp::ResultWifiOff::Includes(_lastResult))
    {
        // 無線オフモード
    }
    else if (nn::nfp::ResultSleep::Includes(_lastResult))
    {
        // スリープ
    }
    else if (nn::nfp::ResultInvalidOperation::Includes(_lastResult))
    {
        // ステートがおかしい
    }
    else
    {
        // そのほかのエラーはPANICで停止させる
        NN_PANIC_WITH_RESULT(_lastResult);
    }
    return _lastResult;
}
```

```
// タグ検知の停止
nn::Result _stopDetection(bool forceDisconnect = false)
{
    // 検知停止
    _lastResult = nn::nfp::StopDetection();
    if (_lastResult.IsSuccess())
    {
        // 処理成功
    }
    else if (nn::nfp::ResultWifiOff::Includes(_lastResult))
    {
        // 無線オフモード
    }
    else if (nn::nfp::ResultSleep::Includes(_lastResult))
    {
        // スリープ
    }
    else if (nn::nfp::ResultInvalidOperation::Includes(_lastResult))
    {
        // ステートがおかしい
    }
    else
    {
        // そのほかのエラーはPANICで停止させる
        NN_PANIC_WITH_RESULT(_lastResult);
    }

    // CTR独自の実装
    if (!nn::os::IsRunOnSnake())
    {
        // 外付けNFCの切断
        if (forceDisconnect && _connected)
        {
            _connected = false;
            _lastResult = nn::nfp::Disconnect();
            if (_lastResult.IsSuccess())
            {
                // 処理成功
            }
            else if (nn::nfp::ResultInvalidOperation::Includes(_lastResult))
            {
                // ステートがおかしい
            }
            else
            {
                // そのほかのエラーはPANICで停止させる
                NN_PANIC_WITH_RESULT(_lastResult);
            }
        }
    }
    return _lastResult;
}
```

```
// NFPライブラリからの通知を待ち受けるスレッド関数
void _procTagDetection()
{
    const int waitMilliSeconds = 500;
    nn::os::WaitObject* events[2] = {
        &_eventActivate, &_eventDeactivate };
    while (!_willTerminate)
    {
        const int eventIndex = nn::os::WaitObject::WaitAny(
            events, 2, nn::fnd::TimeSpan::FromMilliSeconds(waitMilliSeconds));
        if (eventIndex == 0)
        {
            _activate();
        }
        else if (eventIndex == 1)
        {
            _deactivate();
        }
        else
        {
            nn::nfp::TargetConnectionStatus currentTargetConnectionStatus;
            nn::nfp::GetTargetConnectionStatus(&currentTargetConnectionStatus);
            if( _targetConnectionStatus != currentTargetConnectionStatus )
            {
                if (currentTargetConnectionStatus == nn::nfp::TARGET_CONNECTED)
                {
                    _connected = true;
                }
                else if (currentTargetConnectionStatus ==
nn::nfp::TARGET_DISCONNECTED)
                {
                    _connected = false;
                }
                _targetConnectionStatus = currentTargetConnectionStatus;
            }
        }
    }
}
```

6.10.5. システムに負荷がかかっている場合の NFP ライブラリの動作

システムに高い負荷が加わっている場合、NFP ライブラリの使用時に以下のような問題が発生します。

- NFP タグの読み書きが極端に遅くなる
- NFP タグの書き込みに失敗する可能性が高くなる
- NFC リーダー/ライターから切断される
  - NFC リーダー/ライターから切断される現象については、タグが検出されている状態で発生しやすくなっています。NFP ライブラリのステートとしては RW_ACTIVE, RW_MOUNT, RW_MOUNT_ROM の場合にあたります。

そのため、アプリケーション開発時には、以下の点に注意して頂く必要があります。

1. 以下に挙げるデバイスを不必要に使用しないようにする

- カメラ、YUV/RGB 変換、マイク、加速度、ジャイロ、無線、DSP、デバッグパッド(※1)
- 特に、以下のデバイスの使用はシステムコアへの負荷が大きくなりますので、ご注意ください。
  - マイク(高サンプリングレートでの使用)、加速度、無線、デバッグパッド

2. システムコアのアプリへの割り当て時間を不必要に大きくしない
システムコアのアプリへの割り当て時間が減らすことで、1. で挙げたデバイスを使用することによって発生する障害を緩和できる可能性があります。システムコアのアプリでの利用については 「3DS パフォーマンス TIPS」の「システムコアで行われる処理」を参照してください。

3. グラフィックスの描画命令(3D コマンド、コマンドリクエスト)を発行する数を減らす
描画命令の発行数以外にも「3DS パフォーマンス TIPS」の「グラフィックス処理」の項目に書かれている内容を適用することで、システムの負荷を下げることができます。

また、上記とは反対に NFP ライブラリを利用することによって発生するシステムの負荷増大によって、グラフィックスの処理落ちやサウンドノイズが発生することがあります。この現象についても、上記で挙げたデバイスの使用を止めることで改善することがあります。

(※1)デバッグパッドはアプリケーションで使用していなくとも、デバッガに接続しているだけで負荷が発生しますので注意してください。

参考までに、サンプルデモの demo1 をベースに、NFP ライブラリを使用する機能と、ローカル通信、ジャイロセンサ、DSP を使用する機能を追加したサンプルプログラムを使用して、CTR で NFC リーダー/ライターを使用した場合について、NFC リーダー/ライターから切断された頻度について記載します。

このデモでの NFP ライブラリの使用方法は、タグを検出した後 RW_ACTIVE と RW_MOUNT 状態を行き来するだけの使い方になります。そのため、常時 NFC リーダー/ライターからの切断が発生しやすい状態になっていることになります。

有効化したデバイス(使用している設定)	無効化したデバイス	切断頻度(1時間あたり)
DSP(aac)、ローカル通信(2台通信 1Mbps)	カメラ、YUV/RGB 変換、マイク、加速度、ジャイロ、デバッグパッド	0.25回
マイク(SAMPLING_RATE_32730)、DSP(aacdec)、ローカル通信(2台通信 1Mbps)	カメラ、YUV/RGB 変換、加速度、ジャイロ、デバッグパッド	1回
カメラ(FRAME_RATE_30)、DSP(aacdec)、加速度、ジャイロ、ローカル通信(2台通信 1Mbps)	マイク、デバッグパッド	0回
マイク(SAMPLING_RATE_32730)、カメラ(FRAME_RATE_30)、DSP(aacdec)、加速度、ジャイロ	ローカル通信、デバッグパッド	1.5回
マイク(SAMPLING_RATE_8180)、カメラ(FRAME_RATE_30)、DSP(aacdec)、加速度、ジャイロ	ローカル通信、デバッグパッド	0回

demo1 は一般的なアプリケーションと比較してシステムコアに与える負荷が小さいため、実際のアプリケーション開発では上記よりも切断頻度が上がる可能性が高いことに注意してください。

更新履歴

Version 1.4 2016-05-10

変更

- 5.1. 初期化シーケンス
  - 初期化シーケンスのフローにおいて Initialize() が返すエラー値に ResultIrFunctionError を追記しました。

Version 1.3 2015-11-05

変更

- 1. はじめに
  - ほかに赤外線通信を使用中の場合はその機能を先に終了させることについて追記しました。
- 5.7. タグマウントシーケンス
  - nn::nfp::GetTagInfo() が返す可能性のある返り値から ResultTagNotFound を削除しました。
- 5.8. タグ復旧シーケンス
  - nn::nfp::GetTagInfo() が返す可能性のある返り値から ResultTagNotFound を削除しました。
- 5.10.5. アプリ専用領域チェックシーケンス
  - nn::nfp::GetTagInfo() が返す可能性のある返り値から ResultTagNotFound を削除しました。
- 6. NFP ライブラリの利用方法
  - ステートをNONEにさせない場合の注意事項を、SDKの修正により不要となったため削除しました。
- 6.2. タグ検知の開始
  - 高頻度なタグ検知では制限事項があることを追記しました。
- 6.3. タグ情報の取得
  - nn::nfp::GetTagInfo() が返す可能性のある返り値から ResultTagNotFound を削除しました。
- 6.10.4. CTR 対応に必要な実装
  - サンプルコードの誤記を修正しました。

Version 1.2 2015-04-28

追加

- 6.10.5. システムに負荷がかかっている場合の NFP ライブラリの動作

変更

- 2. NFP ライブラリでできること
  - amiibo 以外のタグについて、ニンテンドー 3DS と New ニンテンドー 3DS で検知できるタグに差があることを追記しました。
  - タグを2つ同時に使用できないことと、使用しようとした場合に発生する問題について追記しました。
- 4. ステート遷移
  - NFC リーダー/ライター 使用時のバッテリ持続時間に関する情報を修正しました。
- 5.3. 共通エラーハンドリング
  - 共通のエラーハンドリングに nn::nfp::ResultConnectCanceled を追記しました。
- 5.8. タグ復旧シーケンス
  - nn::nfp::Restore() のエラーに nn::nfp::ResultConnectCanceled を追記しました。
- 5.10.1. 共用情報取得シーケンス
  - nn::nfp::GetNfpCommonInfo() のエラーから nn::nfp::ResultTagNotFound を削除しました。
- 5.10.2. 登録情報取得シーケンス
  - nn::nfp::GetNfpRegisterInfo() のエラーから nn::nfp::ResultTagNotFound を削除しました。
- 5.10.4. アプリ専用領域作成シーケンス

- nn::nfp::CreateApplicationArea() のエラーに nn::nfp::ResultConnectCanceled を追記しました。
- 5.10.6. アプリ専用領域読み込みシーケンス
  - nn::nfp::ReadApplicationArea() のエラーから nn::nfp::ResultTagNotFound を削除しました。
- 5.10.7. アプリ専用領域書き込みシーケンス
  - nn::nfp::Flush() のエラーに nn::nfp::ResultConnectCanceled を追記しました。
- 5.11.3. 電源断シーケンス
  - 電源メニュー表示前に NFP ライブラリを終了する必要があることを追記しました。
- 5.11.4. NFC リーダー/ライター接続シーケンス
  - スリープ状態や無線オフモード遷移によるエラーについて追記しました。
  - ResultConnectCanceled、ResultTimeOutError についてハンドリングを追記しました。
  - フロー図内の nn::nfp::GetConnectResult で得られる失敗要因から ResultInvalidOperation を削除しました。
- 6. NFP ライブラリの利用方法
  - NFP ライブラリの利用について注意事項を追記しました。
  - NFP ライブラリの関数のブロック時間を修正しました。
- 6.1. 初期化
  - NFP ライブラリの関数のブロック時間を修正しました。
  - nn::nfp::Initialize() が返す返り値に nn::nfp::ResultIrFunctionError を追加しました。
- 6.2. タグ検知の開始
  - タグの発見や喪失に要する時間を修正しました。
- 6.4. タグのマウント
  - NFP ライブラリの関数のブロック時間を修正しました。
- 6.4.1. タグの復旧
  - nn::nfp::Restore() が返す返り値について nn::nfp::ResultConnectCanceled を追記しました。
- 6.6.2.1. アプリ専用領域の作成
  - nn::nfp::CreateApplicationArea() が返す返り値に nn::nfp::ResultConnectCanceled を追記しました。
- 6.6.2.2. アプリ専用領域の読み込み
  - nn::nfp::ReadApplicationArea() が返す返り値に nn::nfp::ResultSleep、nn::nfp::ResultWifiOff を追加しました。
- 6.6.2.3. アプリ専用領域への書き込み
  - nn::nfp::Flush() が返す返り値に nn::nfp::ResultConnectCanceled 、nn::nfp::ResultOperationFailed を追記しました。
- 6.8. タグ検知の終了
  - nn::nfp::StopDetection() が返す返り値に nn::nfp::ResultConnectCanceled、nn::nfp::ResultIrFunctionError を追加しました。
- 6.10.1. HOMEボタンへの対処
  - NFP ライブラリを Finalize せずに HOME 遷移しようとした場合の挙動についての説明を修正しました。
  - HOME メニュー遷移時には NFP ライブラリを終了するようにサンプルコードを修正しました。
- 6.10.4. CTR 対応に必要な実装
  - 接続状態取得で得られる値の一覧を追記しました。
  - 失敗要因で得られる値の一覧を追記しました。
  - エラーハンドリングについて「NFC リーダー/ライター接続シーケンス」を参照する補足を追記しました。

Version 1.1 2015-01-15

追加

- 5.11.4. NFC リーダー/ライター接続シーケンス

変更

- 1. はじめに
  - エラーコードリストにエラー時のメッセージ例が記入してあることの案内を追記しました。
- 4. ステート遷移
  - NFC リーダー/ライター 使用時のバッテリ持続時間について調査中であることを追記しました。

- SNAKE のバッテリ持続時間に関する情報を追記しました。
- NFC リーダー/ライター との接続状態から切断された場合のステート遷移について追記しました。

- 5.3. 共通エラーハンドリング
  - ResultInvalidOperation 発生時にユーザーに通知する必要がある点を削除しました。
  - NFC リーダー/ライター 使用時の処理について追記しました。
- 5.4. タグチェックシーケンス
  - Mount() や MountRom() が ResultInvalidFormatVersion を返しうることを追記しました。
- 5.7. タグマウントシーケンス
  - Mount() や MountRom() が ResultInvalidFormatVersion を返しうることを追記しました。
- 5.8. タグ復旧シーケンス
  - Restore() が ResultBackupError を返しうることを追記しました。
- 5.10.2. 登録情報取得シーケンス
  - オーナー登録、ニックネーム登録を必須としないように、説明を修正しました。
- 5.10.3. アプリ専用領域アクセス開始シーケンス
  - アプリ専用領域へのアクセス時に ResultNeedCreate が返った時にユーザーへの通知は不要としました。
- 5.10.5. アプリ専用領域チェックシーケンス
  - OpenApplicationArea() が ResultNotSupported を返す記述を削除しました。
- 5.11.1. HOMEメニュー遷移シーケンス
  - HOMEメニュー遷移時に nfp ライブラリを終了する必要がある点を追記しました。
- 5.11.2. スリープ遷移シーケンス
  - 必要ではない関数呼び出しを削除しました。
- 5.11.3. 電源断シーケンス
  - 電源メニューへの遷移を図に追加しました。
  - 電源メニューへの遷移前にタグの検知を停止する必要があることを追記しました。
- 6. NFP ライブラリの利用方法
  - HOME メニュー、アプレット遷移時には Finalize() が必要であることを追加しました。
  - スリープ遷移時にはタグ検出の停止が明示的に必要であることを削除しました。
- 6.2. タグ検知の開始
  - タグ検知中にタッチパネルを使用できないことを追記しました。
  - SNAKE/CLOSER のタグ認識範囲 (xy±10mm, z+5mm) を追記しました。
- 6.4. タグのマウント
  - table タグの状態と各関数の返り値のタグの状態の説明を修正しました。
  - Mount() や MountRom() が ResultInvalidFormatVersion を返しうることを追記しました。
- 6.4.1. タグの復旧
  - Restore() が ResultBackupError を返しうることを追記しました。
- 6.5.「amiibo設定」の起動
  - amiibo 設定にジャンプしてタグの初期化はできなくなったので関連する記述を修正しました。
- 6.6.2.1. アプリ専用領域の作成
  - 書き込むデータをビッグエンディアンにすることを推奨する文章を変更しました。
- 6.10.1. HOMEボタンへの対処
  - タグ検知中にタッチパネルを使用できないことを追記しました。
  - HOMEメニュー遷移時に nfp ライブラリを終了する必要がある点を追記しました。
- 6.10.2. スリープ要求への対処
  - 必要ではない関数呼び出しを削除しました。
- 6.10.3. 電源ボタンへの対処
  - タグ検知中にタッチパネルを使用できないことを追記しました。
- 6.10.4. CTR 対応に必要な実装
  - 接続状態を確認する方法が変更されたことに伴い、説明を修正しました。
  - 関数の呼び出しにかかる時間を修正しました。
  - 接続状態が切断されるタイムアウト時間を追記しました。
  - SNAKE で各関数を呼び出した場合の動作を追記しました。

- 接続状態を維持することを推奨する注意文を削除しました。
- NFC リーダー/ライターの電池残量を表すLEDの状態を更新する方法について追記しました。

Version 1.0 2014-10-15

追加/変更

- 初版