

SONY®

FeliCa

ソフトウェア

SDK for NFC & Adobe® AIR® / Adobe® Flash® 開発手引書

Sony Corporation Confidential

Version1.3
No. M628-J01-30

はじめに

本文書は、SDK for NFC & Adobe® AIR®/Adobe® Flash®を利用した AIR/Flash アプリケーションの作成方法について説明しています。

- FeliCa は、ソニー株式会社が開発した非接触 IC カードの技術方式です。
- FeliCa は、ソニー株式会社の登録商標です。
- その他、本文書中の会社名や商品名は、該当する各社の商標または登録商標です。
- 本文書の全部または一部の複写、複製および第三者への配布を禁止します。
- 本文書の内容は予告なく変更することがあります。
- 本文書を参照することによって生じた損害について、ソニー株式会社は一切の責任を負いません。
- 本文書によって、記載内容に関する工業所有権の実施許諾や、その他の権利に対する保証を認めたものではありません。
- 本文書中にサンプルソースコードが記載されている場合、これらは使用上の参考として、代表的な応用例を示したものですので、これらサンプルコードの使用に起因する損害について、当社は一切責任を負いません。

(このページは白紙です。)

目次

1. 本文書の目的と用語・参考資料	1
1.1 本文書の目的	1
1.2 用語説明	2
2. SDK for NFC & Adobe® AIR® / Adobe® Flash®を使用して開発を始める前に.....	3
2.1 SDK for NFC & Adobe® AIR® / Adobe® Flash®とは	3
2.2 必要な環境	4
3. SDK for NFC & Adobe® AIR® / Adobe® Flash®の設定方法.....	6
3.1 Adobe Flex Builder 3 の設定方法	6
3.2 Adobe Flash CS4 Professional の設定方法.....	7
4. アプリケーションの開発	13
4.1 おサイフケータイのブラウザを起動させる	13
5. SDK for NFC & Adobe® AIR®/Adobe® Flash®の機能を理解する.....	38
5.1 Basic 版および Standard 版の共通機能.....	38
5.2 Standard 版のみの機能.....	39
6. クラスを理解する	40
6.1 クラス一覧	41
6.2 クラスと機能の対応	43
6.3 イベントの種類	45
7. クラスを活用する	46
7.1 おサイフケータイにアクセスする	46
7.2 NFC フォーラム Tag にアクセスする.....	51
7.3 FeliCa カードを検出する.....	55
7.4 連続して FeliCa カードを検出する	59
7.5 FeliCa カードのデータを読み書きする.....	64
8. アプリケーション開発のためのリソース.....	71
8.1 SDK for NFC and Adobe AIR / Adobe Flash Basic API Documentation (ASDoc)	71
8.2 サンプルプログラム	71
8.3 技術情報（ソニー公式サイト）.....	72
8.4 NFC フォーラム.....	72
8.5 FeliCa Developers' Blog	72

1. 本文書の目的と用語・参考資料

1.1 本文書の目的

本文書は、SDK for NFC & Adobe® AIR® / Adobe® Flash®を利用した AIR アプリケーションまたは Flash アプリケーションの作成方法について説明しています。

SDK for NFC & Adobe® AIR® / Adobe® Flash®は、AIR アプリケーションまたは Flash アプリケーションから FeliCa カードにアクセスするための API を提供します。AIR または Flash アプリケーションは、ActionScript で記述されたスクリプトから、SDK for NFC & Adobe® AIR® / Adobe® Flash®を利用することで、FeliCa カードにアクセスすることができます。

本文書は、SDK for NFC & Adobe® AIR® / Adobe® Flash®を利用した AIR または Flash アプリケーションの作成をする方を、主な読者として想定しています。

1.2 用語説明

本文書で使用されている用語と本文書での意味について、表 1-1 に示します。

表 1-1 : 用語説明

用語	本文書での意味
SDK for NFC & Adobe® AIR® / Adobe® Flash®	NFC デバイスにアクセス（検知、読み書きなど）する機能を持ったライブラリ群。
AIR アプリケーション	アドビシステムズ 社が提供するデスクトップ RIA (Rich Internet Applications) を実現する実行環境を使用したデスクトップアプリケーション。
Flash アプリケーション	アドビシステムズ 社が開発している動画やゲームなどを扱うための規格、およびそれを制作する同社のソフトウェア群を組み合わせで作成した Web コンテンツ。
ActionScript	アドビシステムズ社の製品である Flash に使用されるプログラミング言語。
おサイフケータイ	携帯電話に埋め込まれた FeliCa チップ（IC チップ）を使ったサービス、およびこのサービスに対応したモバイル FeliCa チップを内蔵した携帯電話端末の総称。 ※『おサイフケータイ』は株式会社 NTT ドコモの登録商標です。
NFC フォーラム Tag	NFC フォーラム Tag に準拠した IC タグ。
FeliCa カード	FeliCa 技術方式を有する IC カード、および同等の機能を有する機器。
FeliCa ポートソフトウェア	ソニー株式会社が開発した FeliCa ポート/パソリを利用するために必要な Windows 用ソフトウェア。
NFCProxyService (旧 FeliCaProxyService)	SDK for NFC & Adobe® AIR® / Adobe® Flash®を使用したアプリケーションの実行に必要なソフトウェア。FeliCa ポートソフトウェアに含まれる。
リーダ/ライタ	NFC デバイスに対してデータを読み書きするための機器。

2. SDK for NFC & Adobe® AIR® / Adobe® Flash®を使用して 開発を始める前に

本章では、SDK for NFC & Adobe® AIR® / Adobe® Flash®を使用するにあたって、基本となることを説明します。

2.1 SDK for NFC & Adobe® AIR® / Adobe® Flash®とは

SDK for NFC & Adobe® AIR® / Adobe® Flash®は、AIR アプリケーションまたはFlash アプリケーションから NFC デバイスへのアクセスを行うための API をパッケージ化したクラスライブラリです。

AIR アプリケーションまたはFlash アプリケーションプログラムの ActionScript から SDK for NFC & Adobe® AIR® / Adobe® Flash®の API を呼び出すことで、NFC デバイスにアクセスすることができます。

パッケージ製品の種類

SDK for NFC & Adobe® AIR® / Adobe® Flash®には、2 つのパッケージ製品があります。

1. SDK for NFC & Adobe® AIR® / Adobe® Flash® Standard
2. SDK for NFC & Adobe® AIR® / Adobe® Flash® Basic

以降、それぞれ「Standard 版」、「Basic 版」とします。

パッケージ製品 Standard 版と Basic 版の違い

Basic 版は、おサイフケータイのフリー領域へのアクセス、三者間通信(ブラウザ起動、i アプリ起動)、NFC フォーラム Type2、3、4 Tag へのアクセスと、FeliCa カードを検知する機能を提供しています。

Standard 版は、Basic 版の機能に加えて、FeliCa カードコマンドを実行してカードへ直接アクセスを行うための機能や Edy 読み取り機能が利用可能です。

おサイフケータイの WebTo 機能(URL を送信しブラウザ上で表示する機能)や NFC フォーラム Tag への読み書き、FeliCa カードの検知を行うアプリケーションは、Basic 版で開発できます。

FeliCa カードのデータを読み書きするなどのアプリケーションを開発する場合には、Standard 版をご利用ください。各パッケージが提供する機能については、「5. SDK for NFC & Adobe® AIR®/Adobe® Flash®の機能を理解する」を参照してください。

Basic 版、Standard 版ともに、NDEF フォーマットされた NFC フォーラム Type2、4 Tag へのアクセスのみ可能です。

2.2 必要な環境

SDK for NFC & Adobe® AIR® / Adobe® Flash®を利用したアプリケーションを実行する場合には、以下のソフトウェアが必要です。

- FeliCa ポートソフトウェア Ver. 4.3.2.15 以降
NFC フォーラム Type2、4 Tag にアクセスする場合は、Ver. 4.4.8 以降
- 非接触 IC カードリーダー/ライター (FeliCa ポート/パソリ)
- PC/SC アクティベーター for Type B (Type4b タグにアクセスする場合)
- Adobe® AIR®ランタイム 1.5 以上 (AIR アプリケーションを実行する場合)
- Adobe® Flash® Player 10 以上 (Flash アプリケーションを実行する場合)
- Microsoft .NET Framework 2.0 (FeliCa ポートソフトウェアのバージョンが 4.3.11.2 未満の場合)

また、アプリケーションの開発には以下の開発環境が必要です。

- Flex SDK 3.2 以上

アクセス構成

AIR アプリケーションや Flash アプリケーションで SDK for NFC & Adobe® AIR® / Adobe® Flash®を利用することにより、NFC デバイスへアクセスすることができます (図 2-1)。

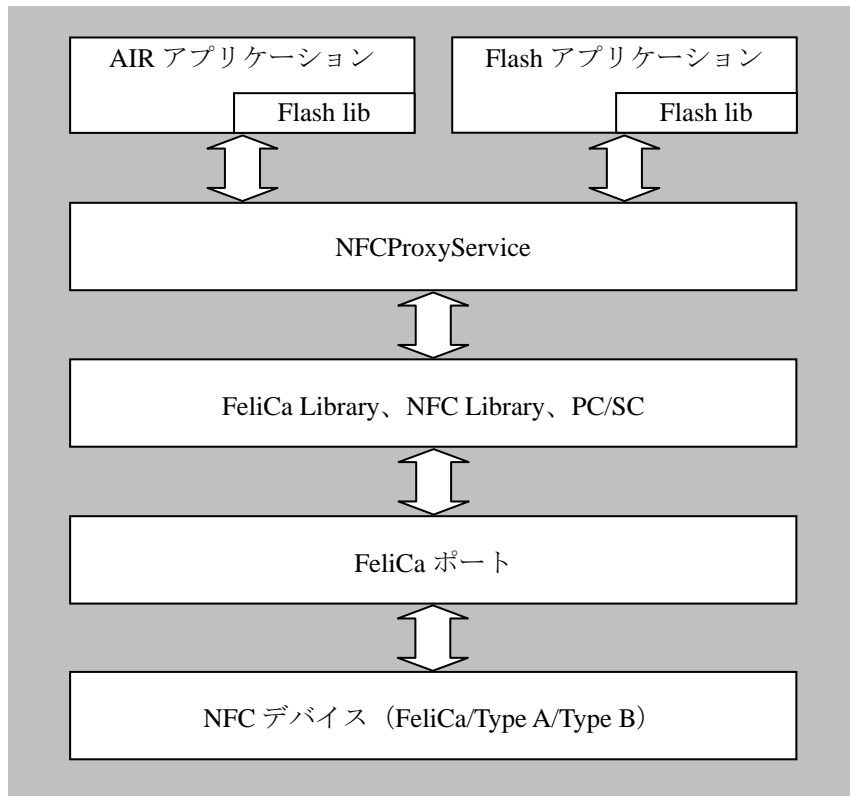


図 2-1 : カードへのアクセス構成

3. SDK for NFC & Adobe® AIR® / Adobe® Flash®の設定方法

本章では、「Adobe Flex Builder 3」および「Adobe Flash CS4」で、AIR /Flash アプリケーションを作成する場合の SDK for NFC & Adobe® AIR® / Adobe® Flash®の導入方法について説明します。

3.1 Adobe Flex Builder 3 の設定方法

SDK for NFC & Adobe® AIR® / Adobe® Flash® のライブラリファイル（Basic 版の場合：SDKforAIR_Flash_Basic.swc、Standard 版の場合：SDKforAIR_Flash_Standard.swc）を開発プロジェクトの「libs」フォルダにコピーします。

Flex Builder 3 は、ライブラリを保存するフォルダとして最初から「libs」フォルダが登録されており、この SWC ファイルをフォルダにコピーしておくだけでライブラリが利用可能になります（図 3-1）。

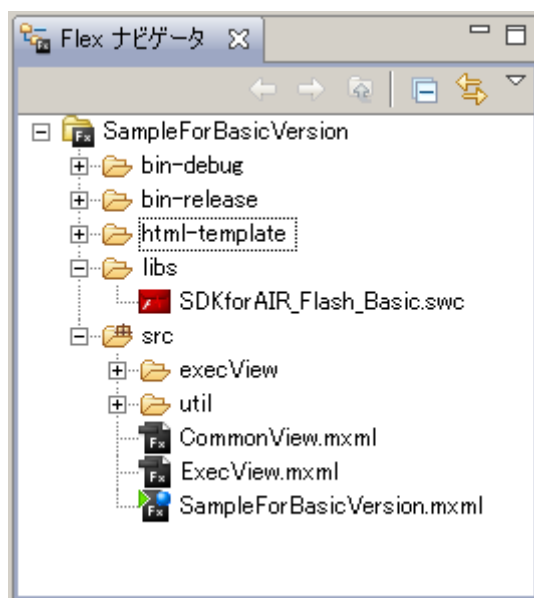


図 3-1 : ライブラリファイルの導入

3.2 Adobe Flash CS4 Professional の設定方法

Adobe Flash CS4 で SDK for NFC & Adobe® AIR® / Adobe® Flash®を使用するには、「ライブラリパス」を設定します。

以下にその手順を示します。

[ファイル] - [パブリッシュ設定] を選択します (図 3-2)。

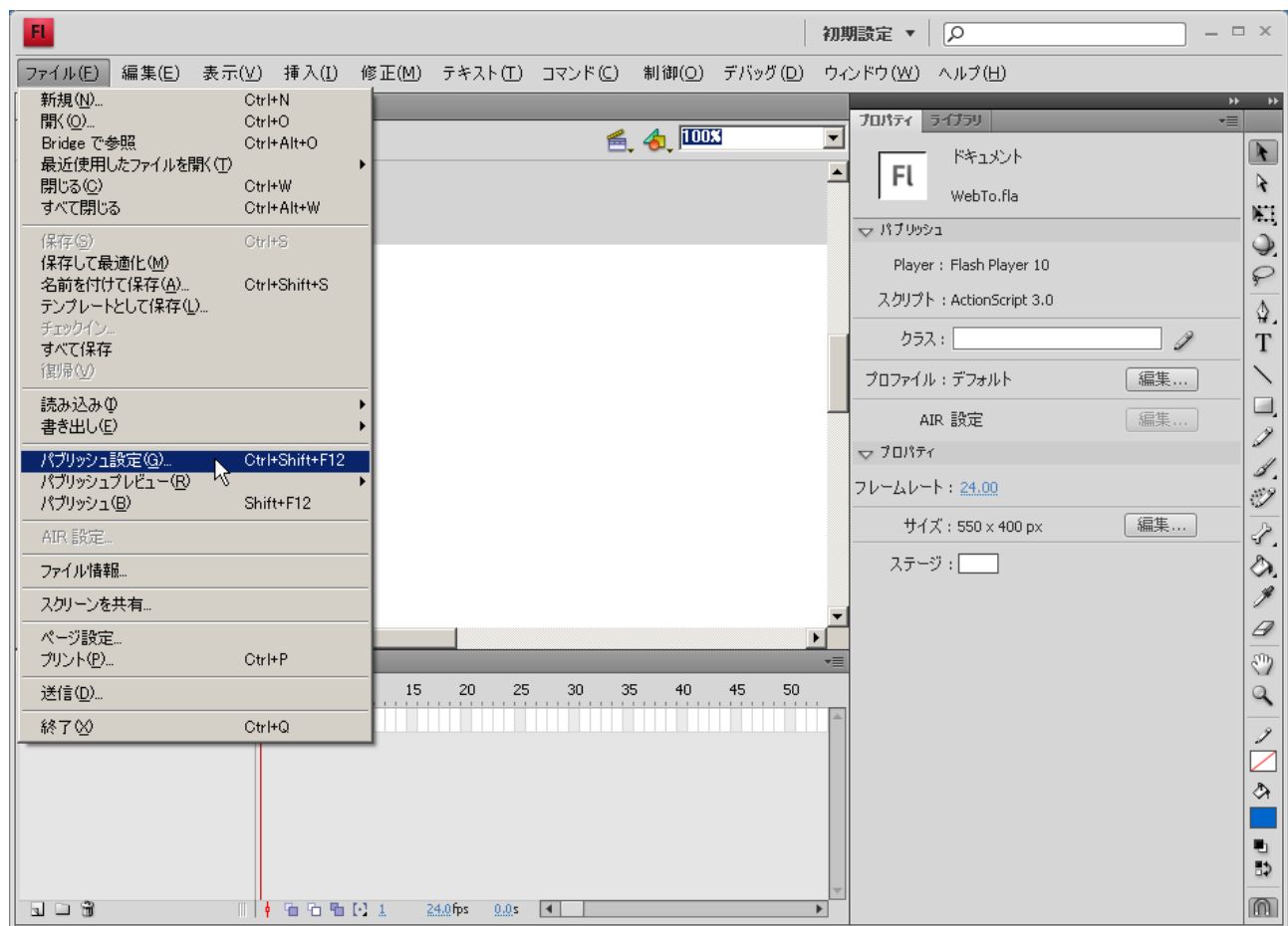


図 3-2 : [ファイル] - [パブリッシュ設定]を選択

表示されたパブリッシュ設定ダイアログの **[Flash]** タブをクリックし、「スクリプト」を「ActionScript 3.0」を選択して、「設定」ボタンをクリックしてください（図 3-3）。

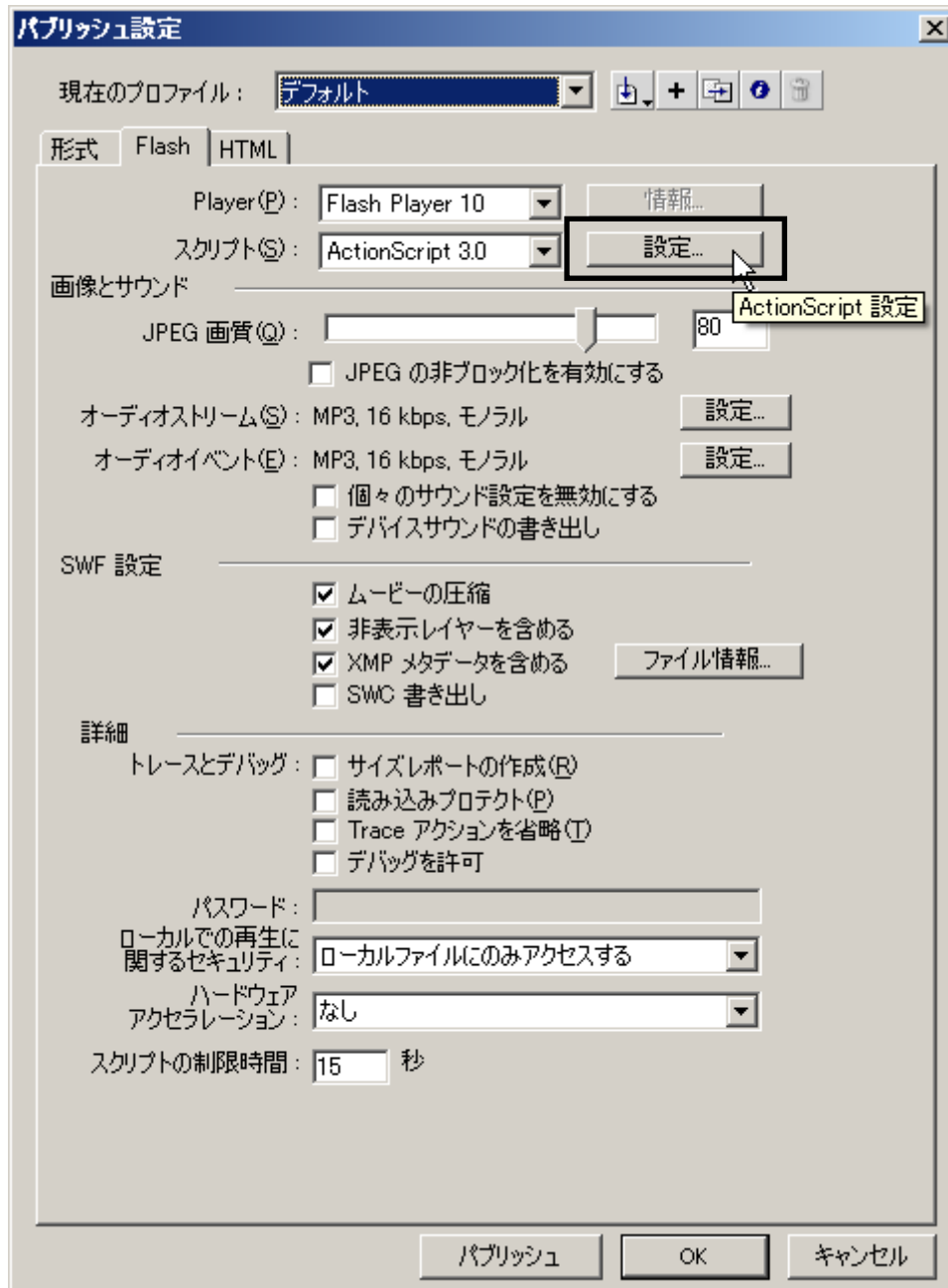


図 3-3 : ActionScript の設定

表示された詳細設定画面の[ライブラリパス]タブをクリックし、**+** ボタンをクリックして新規パスの追加を行います（図 3-4）。

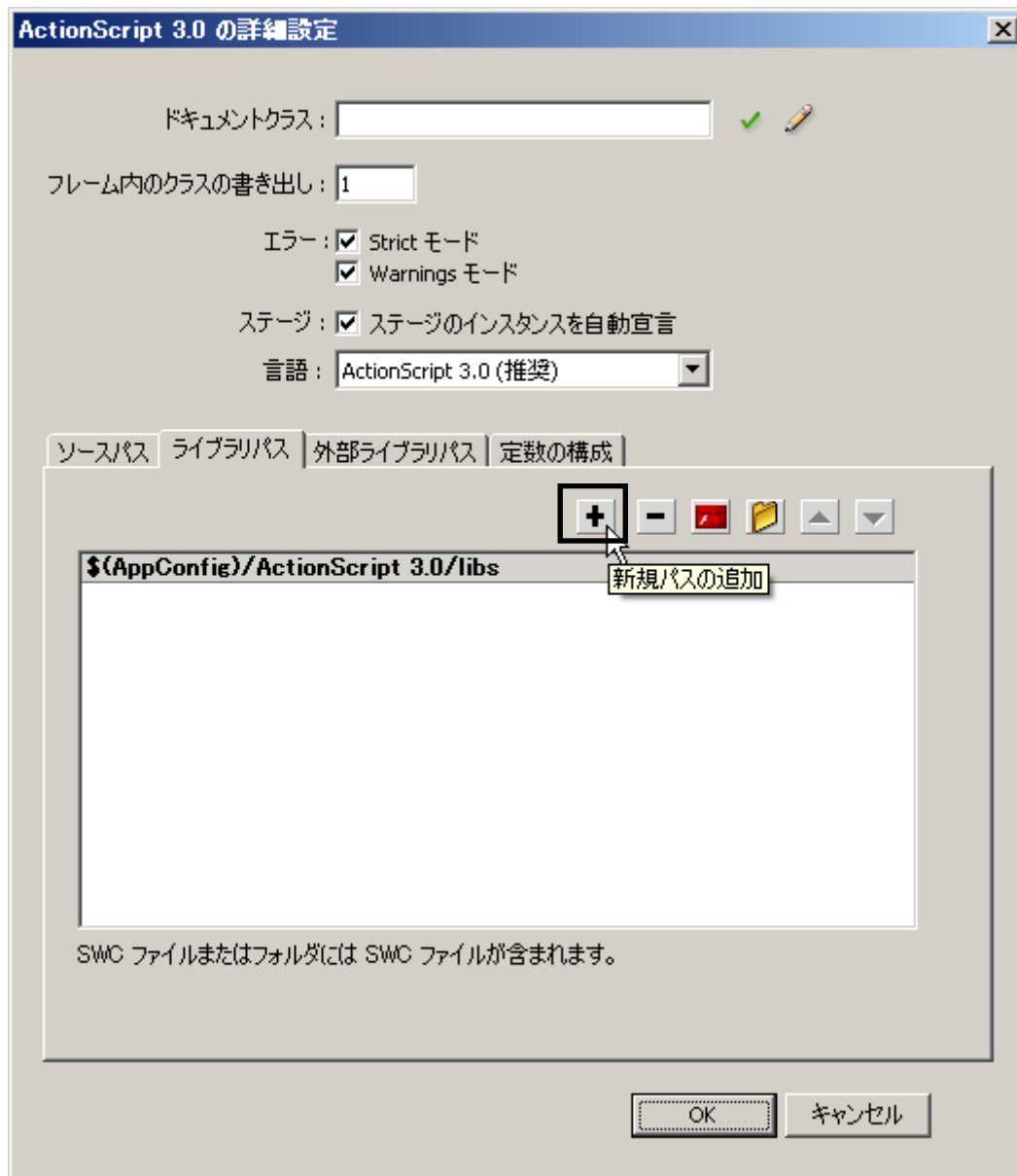



図 3-4 : 新規パスの追加

先程追加した新規パスを選択した状態で、をクリック（図 3-5）すると、ファイル選択ダイアログが表示されます（図 3-6）。SDK for NFC & Adobe® AIR® / Adobe® Flash®のライブラリファイル（Basic 版の場合：SDKforAIR_Flash_Basic.swc、Standard 版の場合：SDKforAIR_Flash_Standard.swc）を選択して、「開く」をクリックしてください。

ライブラリパスを指定した後に「OK」ボタンをクリックして、ActionScript 3.0 の詳細設定画面を終了します。

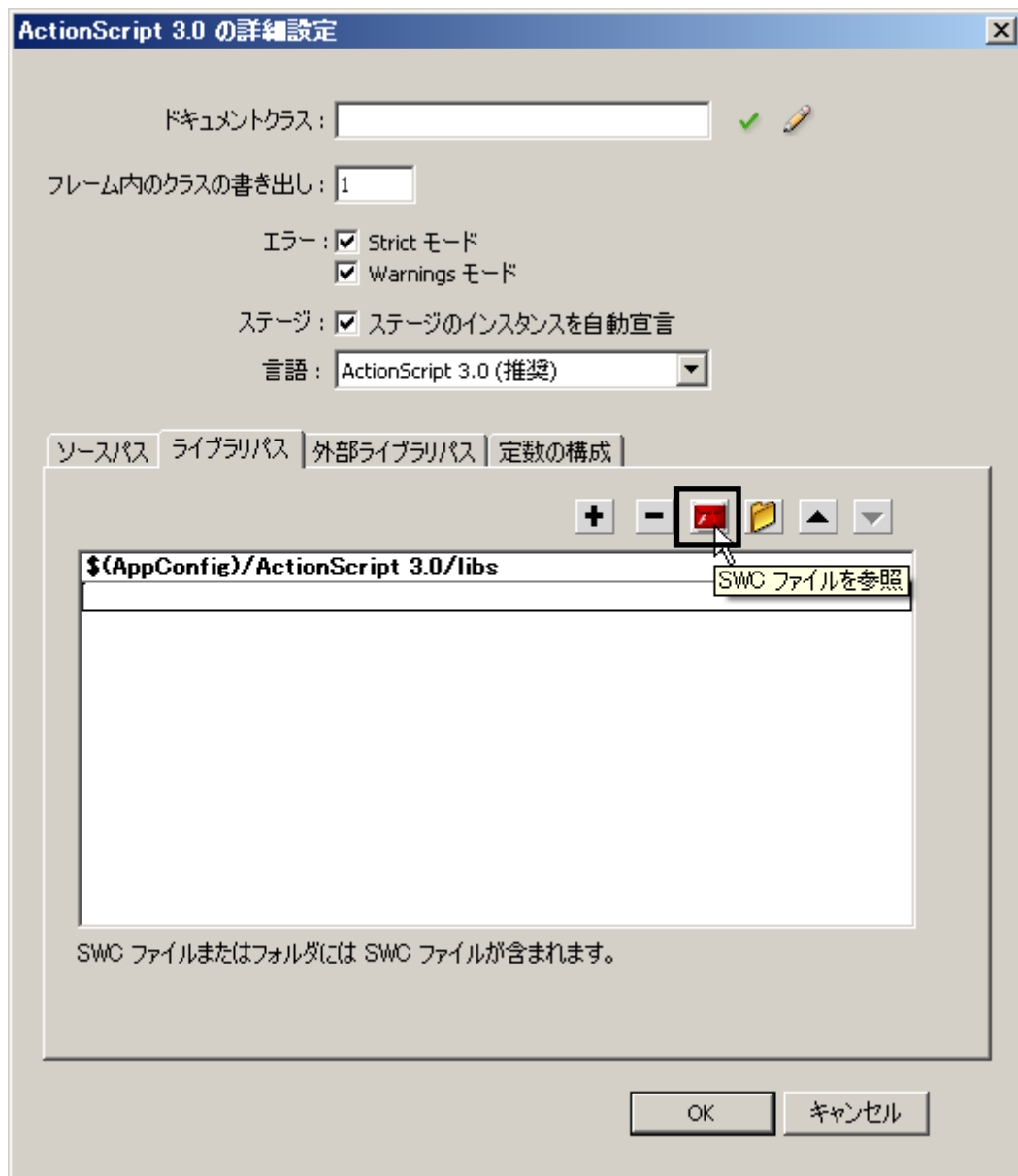


図 3-5 : SWC ファイルの参照

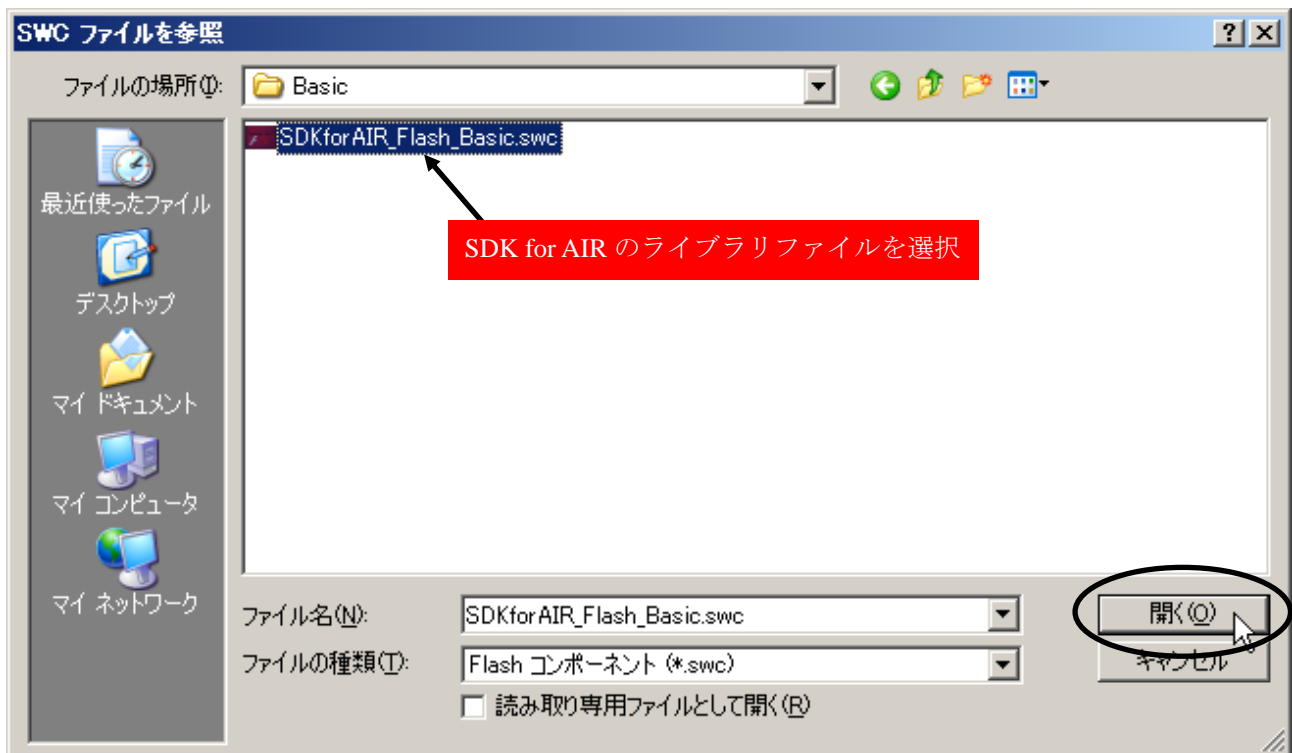


図 3-6 : ファイルダイアログ

パブリッシュ設定画面「詳細」の「ローカルでの再生に関するセキュリティ」で「ネットワークのみにアクセスする」を選択して、「OK」ボタンでパブリッシュ設定画面を終了します（図 3-7）。

※NFCProxyService とネットワーク通信をするため、この設定が必要です。

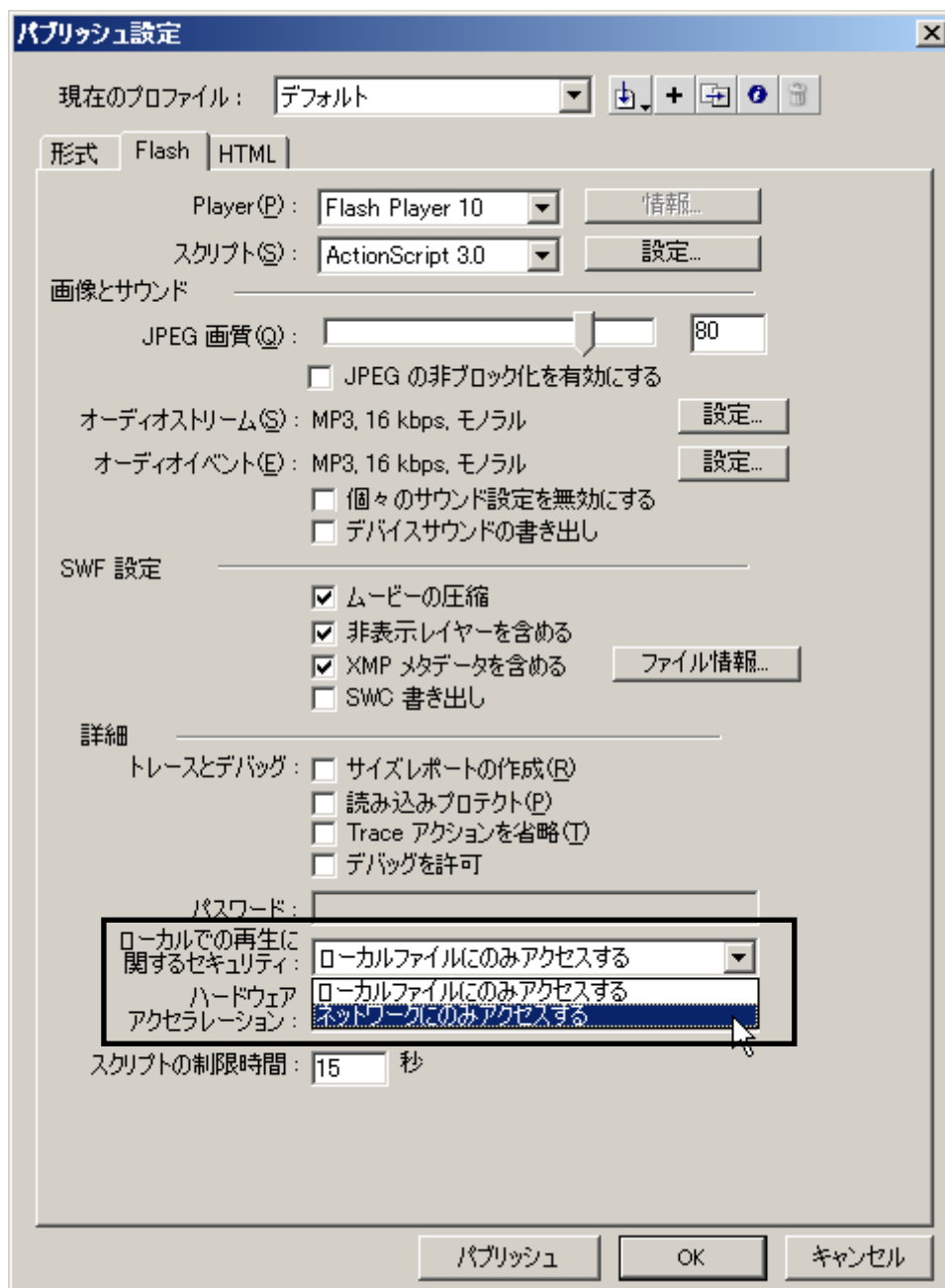


図 3-7：ローカルでの再生に関するセキュリティ

4. アプリケーションの開発

本章では、SDK for NFC & Adobe® AIR® / Adobe® Flash®の基本的な手順を理解するために、簡単なFlashアプリケーションの作成方法を説明します。

SDK for NFC & Adobe® AIR® / Adobe® Flash®を使った処理の流れは、以下のようになります。

1. NFCProxyService と認証
2. NFC デバイスへのアクセスの実行
3. NFCProxyService を切断

※サンプルプロジェクト名：WebTo

4.1 おサイフケータイのブラウザを起動させる

簡単なFlashアプリケーションとして、ボタンをクリックするとおサイフケータイのブラウザを起動させるFlashアプリケーションをAdobe Flex Builder 3（以降、Flex Builder とします）で作成します。

動作は、ボタンをクリックすると以下の処理を行うようにします。

1. NFCProxyService と認証を行う
2. 「おサイフケータイのブラウザを起動させる」API を実行する
3. NFCProxyService との認証を切断する

■ プロジェクトを作る

Flex Builder を起動し、[ファイル] - [新規] - [Flex プロジェクト] を選択します (図 4-1)。

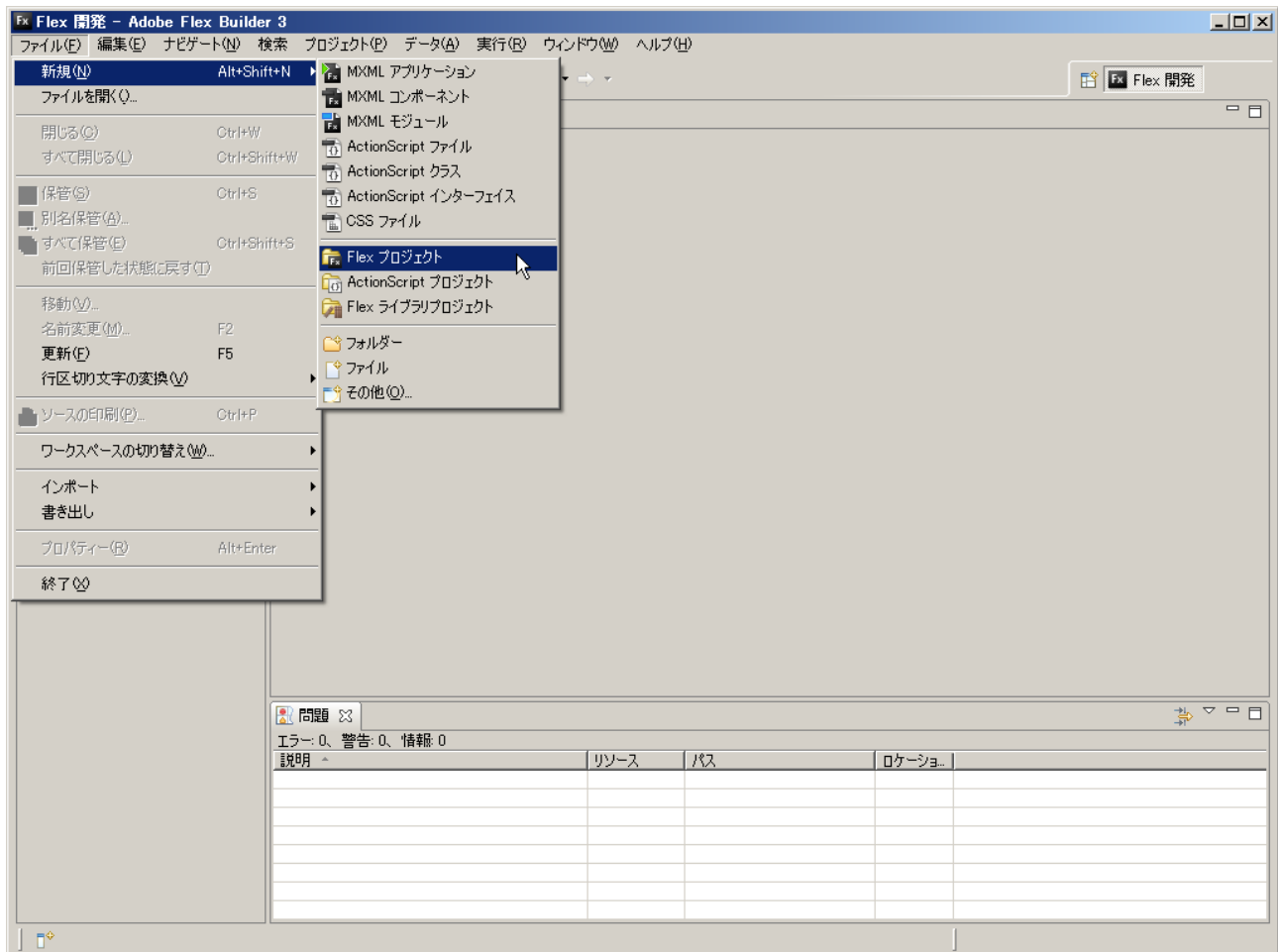


図 4-1 : [ファイル] → [新規] → [Flex プロジェクト]を選択

表示されたプロジェクト作成ウィザード画面で、プロジェクト名とアプリケーションの種別を設定します。
ここでは、プロジェクト名に「WebTo」、アプリケーション種別は「Web アプリケーション」とします(図 4-2)。

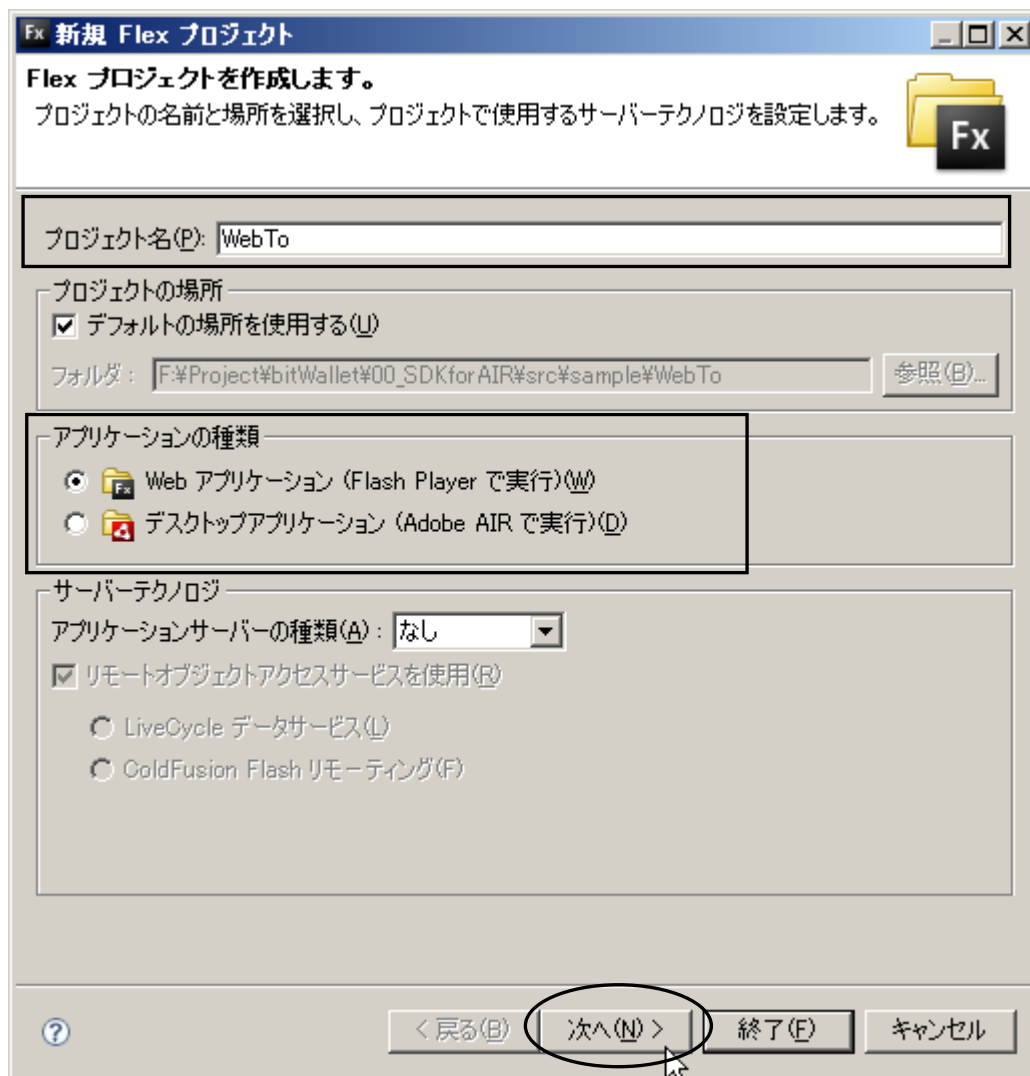


図 4-2：プロジェクト名とアプリケーション種別の設定

プロジェクト名とアプリケーション種別を設定して、[次へ] ボタンをクリックすると、出力設定ウィザード画面が表示されます。出力フォルダを確認し、[次へ] ボタンをクリックします（図 4-3）。



図 4-3 : 出力設定

プロジェクトのビルドパスなどの設定ウィザードが表示されます。設定内容を確認し、[終了] ボタンをクリックします (図 4-4)。

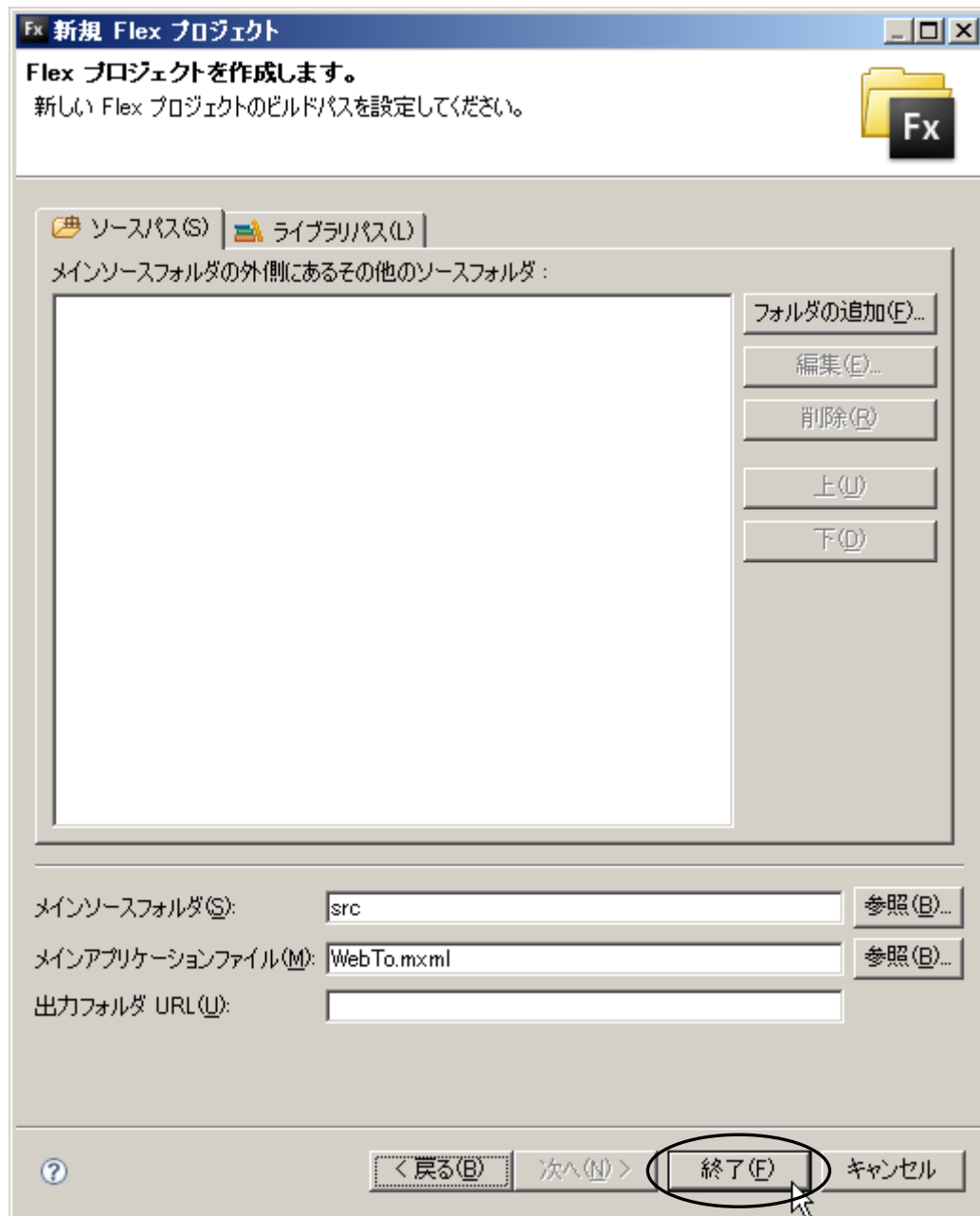


図 4-4 : ビルドパスなどの設定

Flex プロジェクトの開発画面が表示されます（図 4-5）。

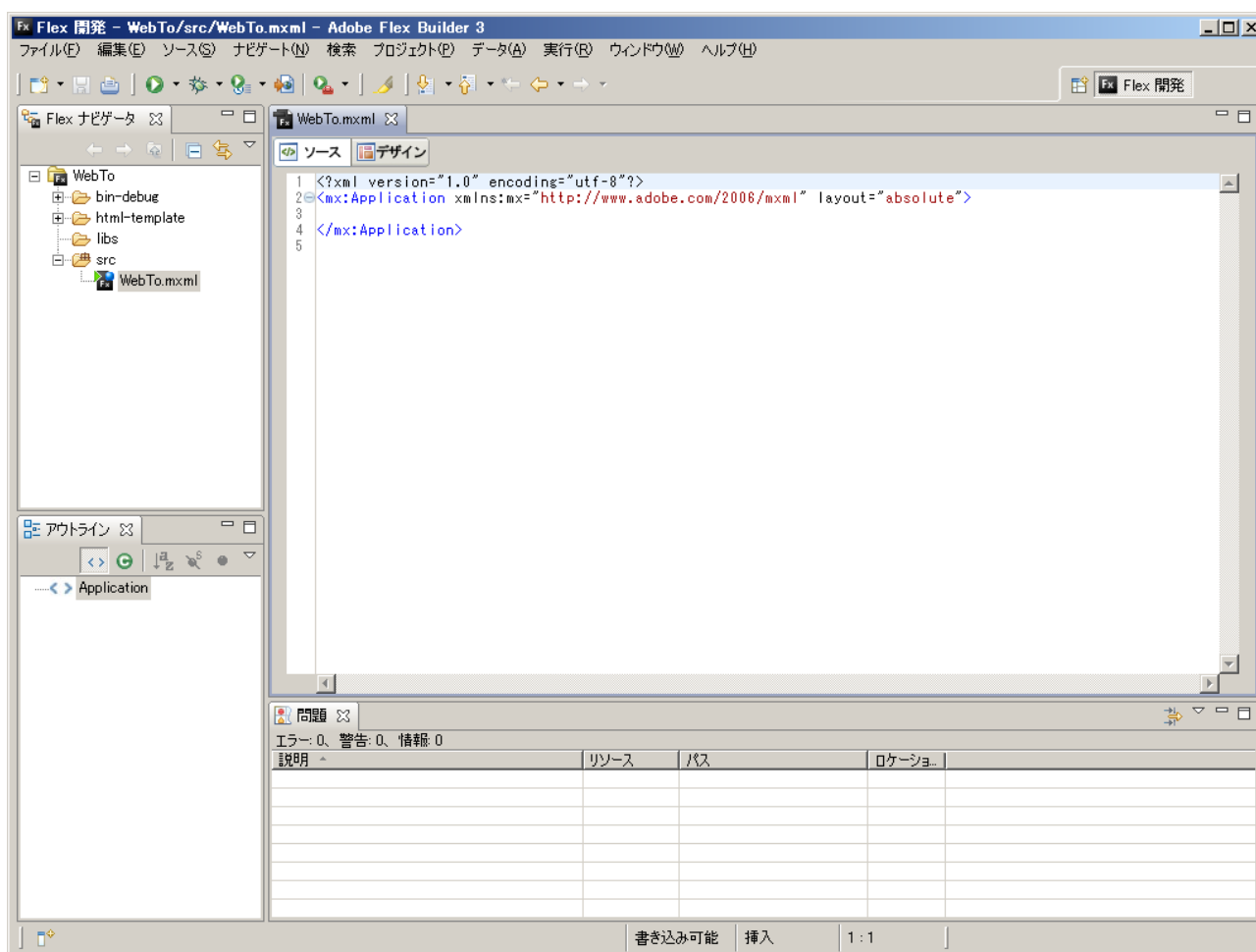


図 4-5 : Flex プロジェクト開発画面

■SDK for NFC & Adobe® AIR® / Adobe® Flash® ライブラリファイルの追加

SDK for NFC & Adobe® AIR® / Adobe® Flash®のライブラリファイルを利用できるようにするため、SDK for NFC & Adobe® AIR® / Adobe® Flash®のライブラリファイル（Basic 版の場合：SDKforAIR_Flash_Basic.swc、Standard 版の場合：SDKforAIR_Flash_Standard.swc）をプロジェクトの「libs」フォルダにコピーします（図 4-6）。

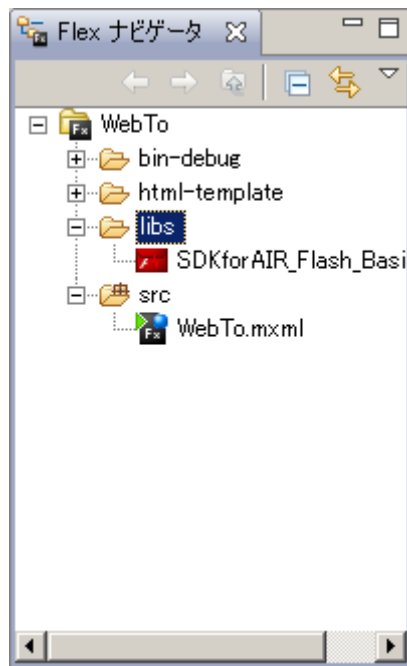


図 4-6 : SDK for AIR の追加

■ ボタンを配置する

ボタンを配置するために、デザインビューに切り替えます。

Flex プロジェクト開発画面の中央部にある切り替えタブで、[デザイン]を選択します（図 4-7）。

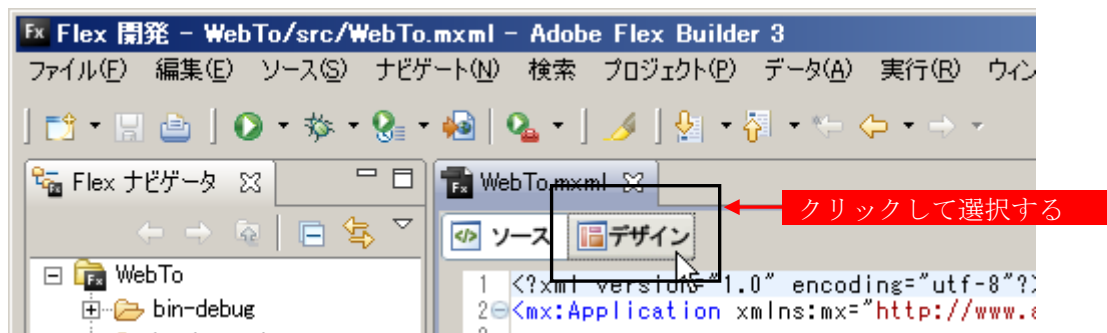


図 4-7：デザインビューに切り替え

コンポーネントパネルからボタンを選択して、ドラッグ&ドロップでボタンを配置します（図 4-8）。

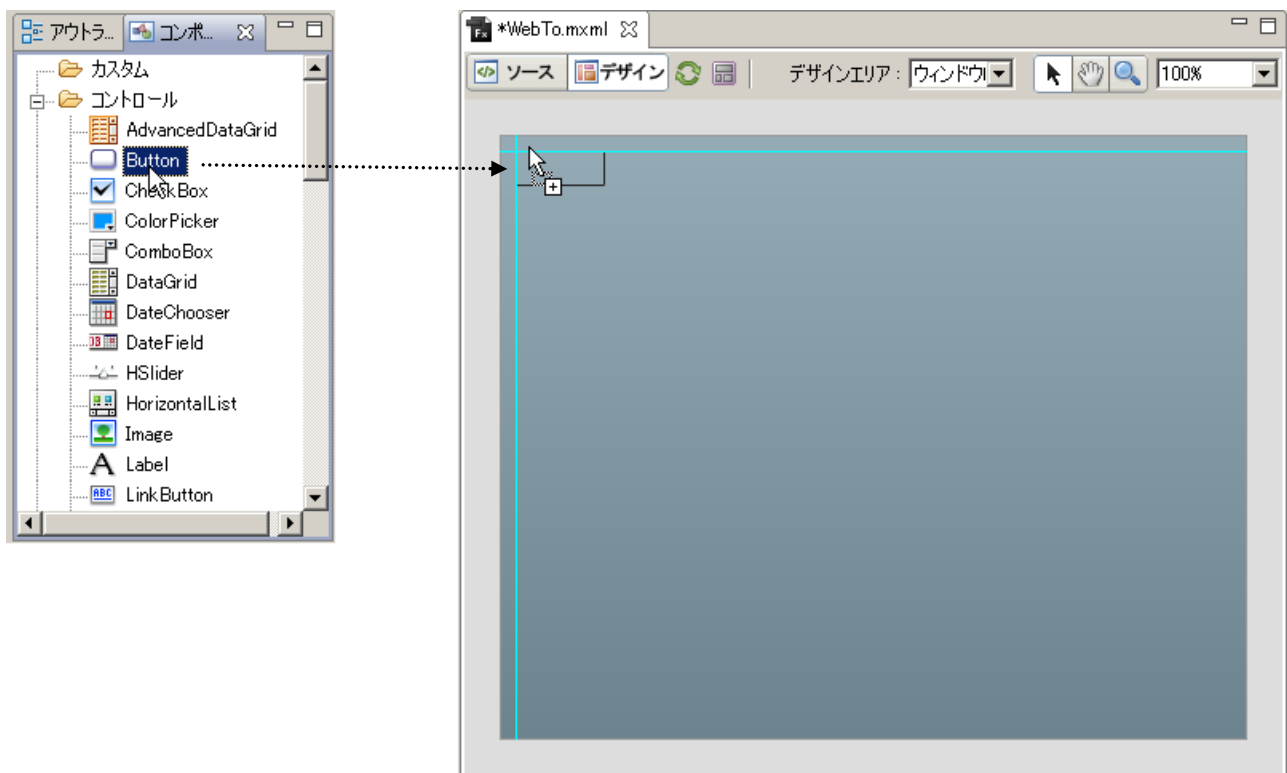


図 4-8：ドラッグ&ドロップでボタンを配置

■ ボタンの ID とラベルを設定する

配置したボタンを選択して、プロパティパネルで ID とラベルを設定します。

ここでは、ID を「button1」、ラベルを「ブラウザ起動」とします（図 4-9）。

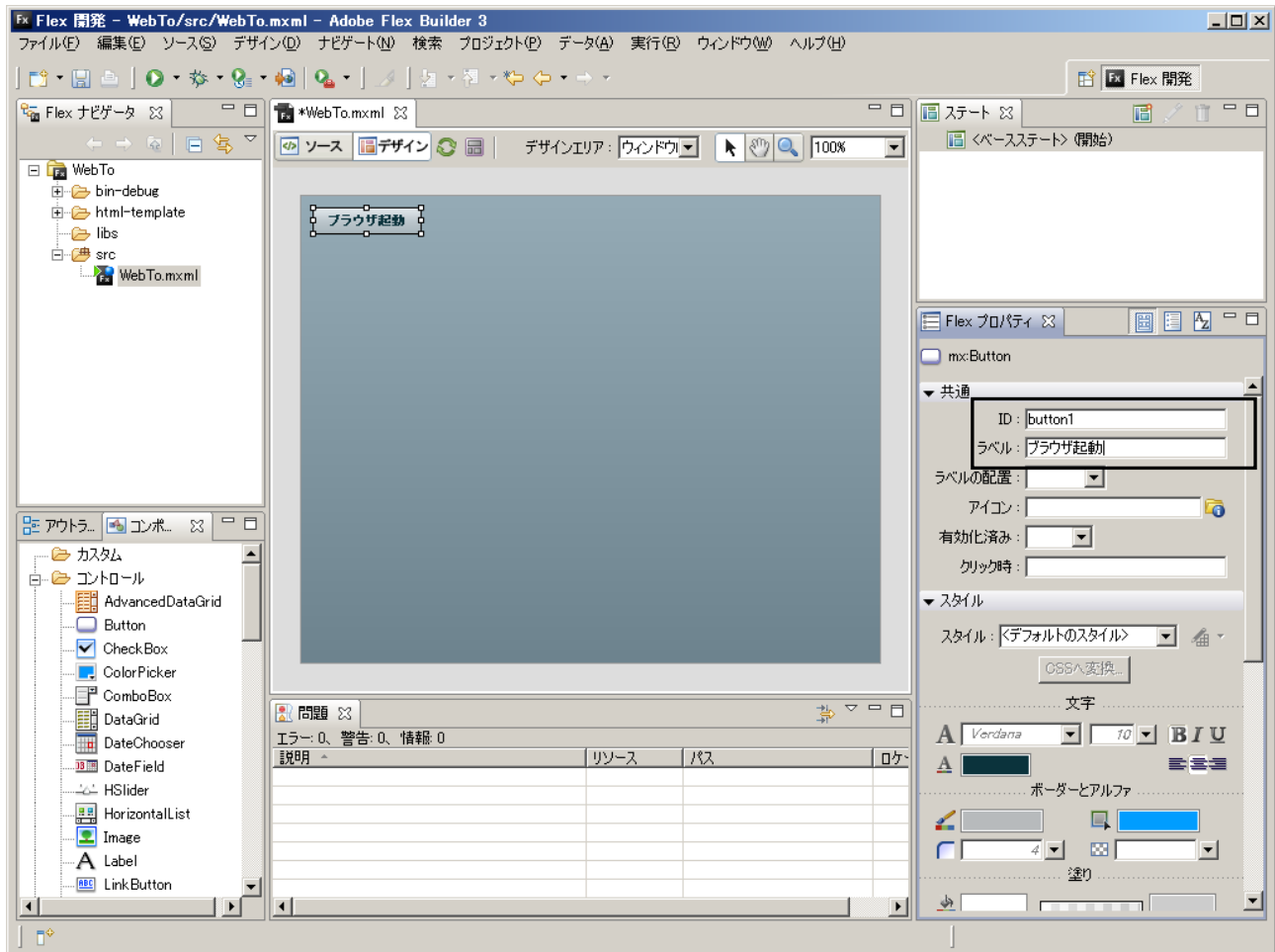


図 4-9 : ID とラベルの設定

■ ボタンクリックに呼ばれる関数の指定

ID とラベルを設定した時と同様に配置したボタンを選択し、プロパティパネルでクリック時を指定します。ここでは、「buttonClick();」とします（図 4-10）。

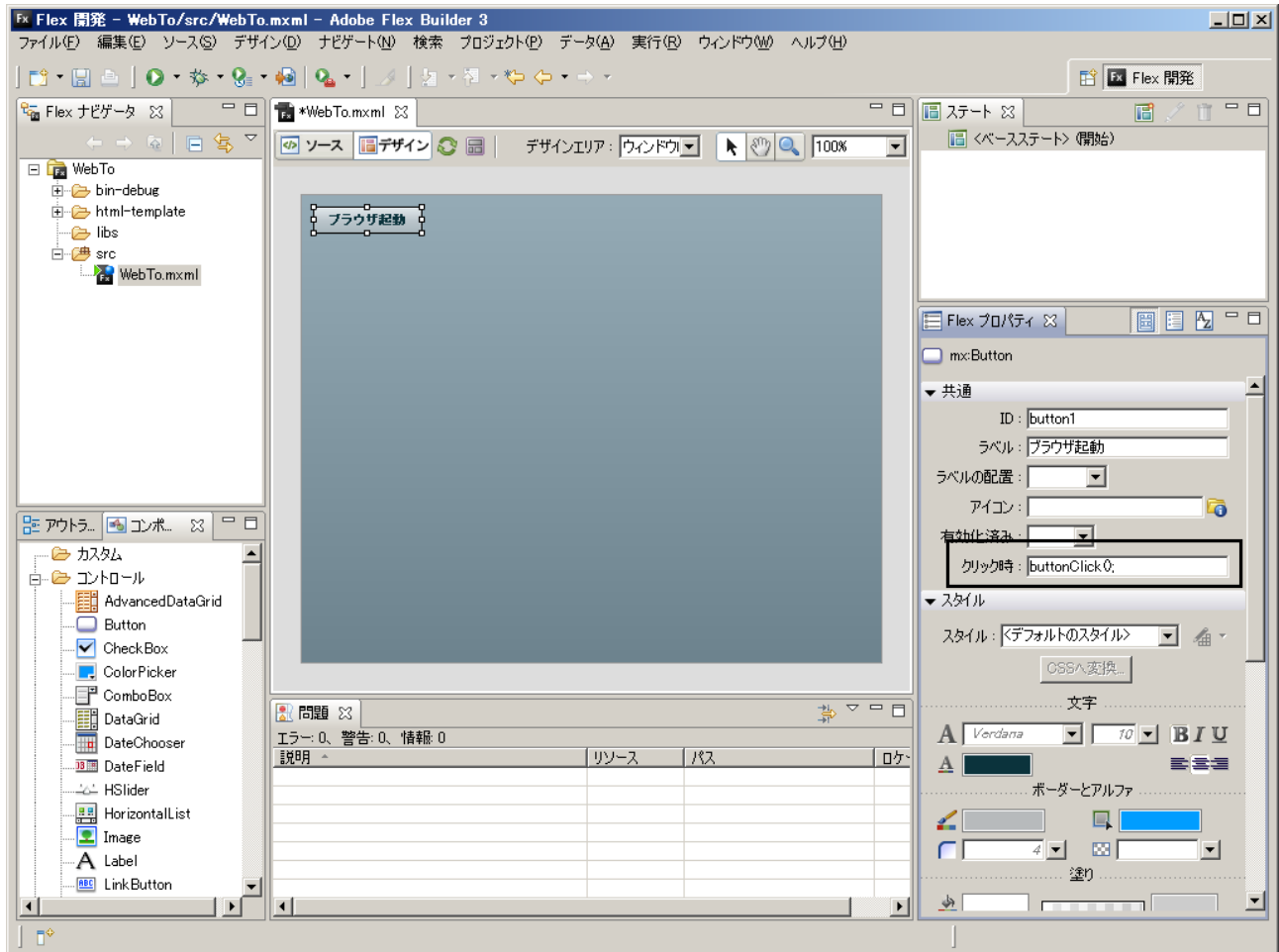


図 4-10 : クリック時に呼ばれるメソッドの設定

■ ボタンクリックに呼ばれる関数を定義する

関数を定義するために、ソースビューに切り替えます。

ボタンを配置した時のように、Flex プロジェクト開発画面の中央部にある切り替えタブで、[ソース]を選択します（図 4-11）。

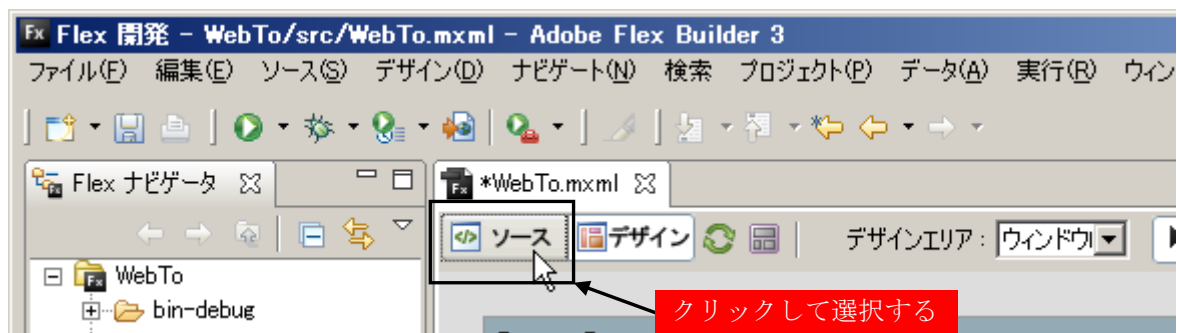


図 4-11 : ソースビューに切り替え

関数の定義は、「mx:Script」タグの内部で定義します。

「mx:Script」タグは、「mx:Application」タグ内であればどこに書いても構いません。

ここでは、「mx:Button」タグの後に書くことにします。

図 4-12 の黒枠部のコードを「mx:Button」タグの後に入力してください。



図 4-12 : 関数を定義

ここで、いったんプロジェクトを保存しておきます（図 4-13）。

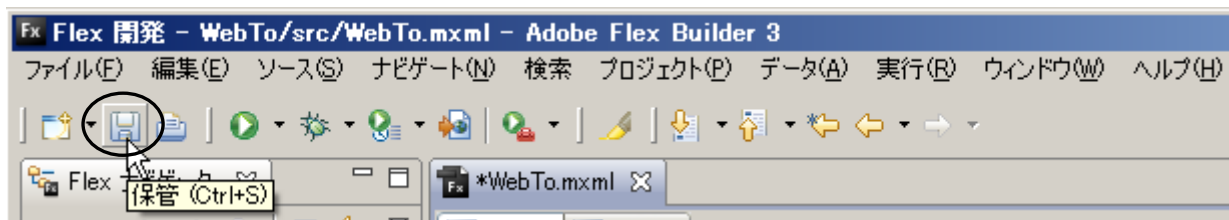


図 4-13 : プロジェクトの保管

■ NFCProxyService と認証する

まずは、ボタンをクリックされた時に最初の処理「NFCProxyService と認証を行う」を記述することになります。

NFCProxyService と認証するために、必要なパッケージをインポートします。

今回必要なパッケージは、以下の 2 つです。

- com.sony.jp.felica.event.OpenStatusEvent
- com.sony.jp.felica.FeliCaControl

パッケージの import 文を、先程定義した buttonClick 関数の前に入力して（図 4-14 黒枠部）、保存します。

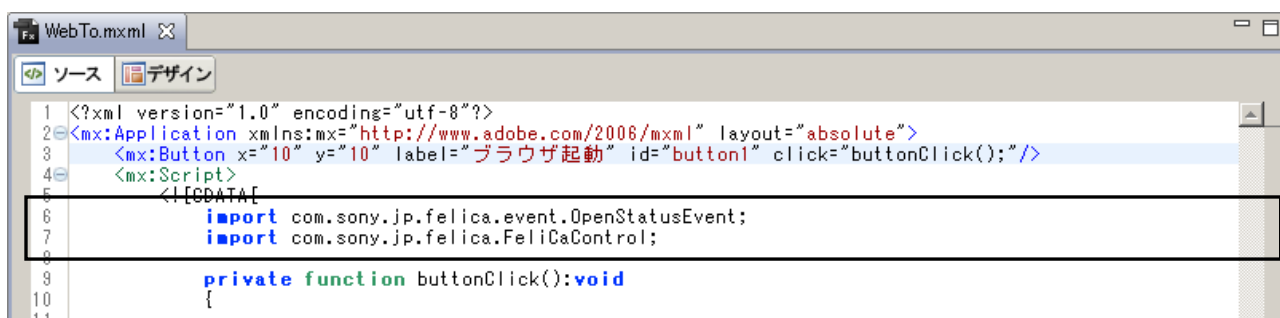
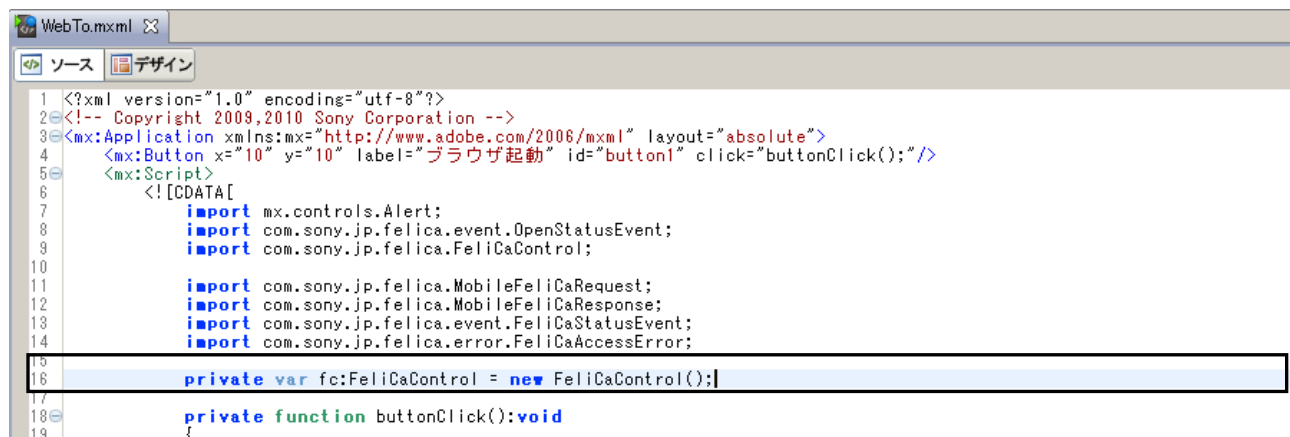


図 4-14 : パッケージのインポート

NFCProxyService と通信を行うには、FeliCaControl クラスを使用します。

「NFCProxyService と認証」や「おサイフケータイのブラウザ起動させる」などの処理を行う FeliCaControl クラスを宣言します。

FeliCaControl クラスを宣言する図 4-15 の黒枠部のコードを入力して、保存します。



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!-- Copyright 2009,2010 Sony Corporation -->
3 <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
4   <mx:Button x="10" y="10" label="ブラウザ起動" id="button1" click="buttonClick();" />
5   <mx:Script>
6     <![CDATA[
7       import mx.controls.Alert;
8       import com.sony.jp.felica.event.OpenStatusEvent;
9       import com.sony.jp.felica.FeliCaControl;
10
11       import com.sony.jp.felica.MobileFeliCaRequest;
12       import com.sony.jp.felica.MobileFeliCaResponse;
13       import com.sony.jp.felica.event.FeliCaStatusEvent;
14       import com.sony.jp.felica.error.FeliCaAccessError;
15
16       private var fc:FeliCaControl = new FeliCaControl();
17
18       private function buttonClick():void
19     {
```

図 4-15 : FeliCaControl クラスの宣言

NFCProxyService と認証するには、FeliCaControl クラスの open 関数を使用します。

open 関数の第一引数として、NFCProxyService の TCP/IP 通信用ポート番号「10250」（固定）を指定します。Standard 版では、open の第二引数にライセンスキーを指定する必要があります(第三引数には空文字を指定してください)。

NFCProxyService と認証するコード(図 4-16 の黒枠部)を buttonClick 関数の中に入力して、保存します。



図 4-16 : NFCProxy の認証

NFCProxyService との認証結果はイベントにて通知されるため、結果を受け取るイベントリスナーを登録します。

イベントの種類を表 4-1 に示します。

表 4-1 : NFCProxyService 認証時のイベントの種類

No	クラス	イベントタイプ定数	説明
1	OpenStatusEvent	OPEN_COMPLETE	NFCProxyService との認証に成功した時に送出される
2	OpenStatusEvent	OPEN_FAILURE	NFCProxyService との認証に失敗した時に送出される

ここでは、イベントを通知するメソッドを「認証に成功した」場合には onOpenComplete、「認証に失敗した」場合には onOpenFailure とします。

NFCProxyService との認証結果を受け取るイベントリスナーを登録するコード(図 4-17 の黒枠部)を入力して、保存します。


```
6      <![CDATA[
7          import mx.controls.Alert;
8          import com.sony.jp.felica.event.OpenStatusEvent;
9          import com.sony.jp.felica.FelicaControl;
10
11          import com.sony.jp.felica.MobileFelicaRequest;
12          import com.sony.jp.felica.MobileFelicaResponse;
13          import com.sony.jp.felica.event.FelicaStatusEvent;
14          import com.sony.jp.felica.error.FelicaAccessError;
15
16          private var fc:FelicaControl = new FelicaControl();
17
18          private function buttonClick():void
19          {
20              fc.addEventListener(OpenStatusEvent.OPEN_COMPLETE, onOpenComplete);
21              fc.addEventListener(OpenStatusEvent.OPEN_FAILURE, onOpenFailure);
22
23              fc.open(10250);
24          }
25
26          private function onOpenComplete(evt:OpenStatusEvent):void
27          {
28          }
29
30          private function onOpenFailure(evt:OpenStatusEvent):void
31          {
32          }
33      ]>
34  
```

図 4-17 : NFCProxyService との認証用イベントリスナー登録

NFCProxyService との認証に失敗した場合には、エラー情報をアラート (Alert) で表示するようにします。

図 4-18 の黒枠部のコードを onOpenFailure 関数の中に入力して、保存します。

```
16 private var fc:FeliCaControl = new FeliCaControl();
17
18 private function buttonClick():void
19 {
20     fc.addEventListener(OpenStatusEvent.OPEN_COMPLETE, onOpenComplete);
21     fc.addEventListener(OpenStatusEvent.OPEN_FAILURE, onOpenFailure);
22
23     fc.open(10250);
24 }
25
26 private function onOpenComplete(evt:OpenStatusEvent):void
27 {
28
29 }
30
31 private function onOpenFailure(evt:OpenStatusEvent):void
32 {
33     var error:Error = evt.object as Error;
34     var msg:String = "NFC Proxy との相互認証に失敗しました。¥n" +
35                     "Error: (" + error.errorID + ") " + error.message;
36     mx.controls.Alert.show(msg);
37 }
38
```

図 4-18 : NFCProxyService との認証エラー情報の表示

■ おサイフケータイのブラウザ起動させる

次の処理として「おサイフケータイのブラウザ起動させる」を記述します。

NFCProxyService の認証時と同様に、まず、必要な以下 4 パッケージをインポートします。

- com.sony.jp.felica.MobileFeliCaRequest
- com.sony.jp.felica.MobileFeliCaResponse
- com.sony.jp.felica.event.FeliCaStatusEvent
- com.sony.jp.felica.error.FeliCaAccessError

図 4-19 の黒枠部のコードを入力して、保存します。

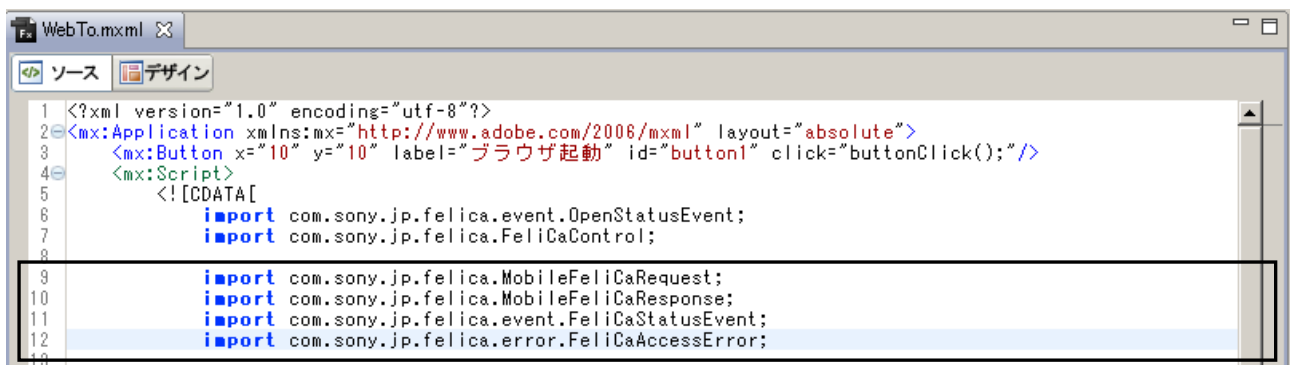


図 4-19 : パッケージのインポート

おサイフケータイのブラウザ起動をさせるには、ブラウザ起動情報を設定する必要があります。

ブラウザ起動情報を設定するには、MobileFeliCaRequest クラスを用います。

ブラウザ起動時の MobileFeliCaRequest クラスの設定を、表 4-2 に示します。

表 4-2 : ブラウザ起動の設定

No	プロパティ	説明
1	type	おサイフケータイへのアクセス方法を設定します。 ブラウザ起動時は、MobileFeliCaRequest.WEBTO を指定します。
2	url	ブラウザ起動時の URL を設定します。

ここでは、ブラウザ起動時の URL を「http://mobile.sony.jp/」として、おサイフケータイのブラウザ起動させる図 4-20 の黒枠部のコードを、NFCProxyService との認証が成功時のイベントハンドラに入力して、保存します。

```
23         fc.open(10250);
24     }
25
26     private function onOpenComplete(evt:OpenStatusEvent):void
27     {
28         fc.addEventListener(FeliCaStatusEvent.FELICA_ACCESS_COMPLETE, onFeliCaAccessComplete);
29         fc.addEventListener(FeliCaStatusEvent.FELICA_ACCESS_FAILURE, onFeliCaAccessFailure);
30         fc.addEventListener(FeliCaStatusEvent.FELICA_ACCESS_PARAMETER_ERROR, onFeliCaAccessParameterError);
31
32         var request:MobileFeliCaRequest = new MobileFeliCaRequest();
33         request.type = MobileFeliCaRequest.WEBTO;
34         request.url = "http://mobile.sony.jp/";
35         fc.access(request);
36     }
37
38     private function onOpenFailure(evt:OpenStatusEvent):void
39     {
40         var error:Error = evt.object as Error;
```

図 4-20 : おサイフケータイのブラウザ起動

NFCProxyService の認証時と同様に、結果を受け取るイベントリスナーを登録します。

イベントの種類を表 4-3 に示します。

表 4-3 : FeliCa カードへアクセス中のイベントの種類

No	クラス	イベントタイプ定数	説明
1	FeliCaStatusEvent	FELICA_ACCESS_COMPLETE	要求が成功した時に送出される
2	FeliCaStatusEvent	FELICA_ACCESS_FAILURE	エラーが発生した時に送出される
3	FeliCaStatusEvent	FELICA_ACCESS_PARAMETER_ERROR	パラメータに不備がある時に送出される

ここでは、イベントを通知するメソッドを「ブラウザ起動が成功」した場合には onFeliCaAccessComplete、「ブラウザ起動が失敗」した場合には onFeliCaAccessFailure、「パラメータ設定エラー」の場合には onFeliCaAccessParameterError とします。

おサイフケータイのブラウザ起動結果を受け取るイベントリスナーを登録するコード(図 4-21 の黒枠部)を入力して、保存します。

```

26 private function onOpenComplete(evt:OpenStatusEvent):void
27 {
28     fc.addEventListener(FeliCaStatusEvent.FELICA_ACCESS_COMPLETE, onFeliCaAccessComplete);
29     fc.addEventListener(FeliCaStatusEvent.FELICA_ACCESS_FAILURE, onFeliCaAccessFailure);
30     fc.addEventListener(FeliCaStatusEvent.FELICA_ACCESS_PARAMETER_ERROR, onFeliCaAccessParameterError);
31
32     var request:MobileFeliCaRequest = new MobileFeliCaRequest();
33     request.type = MobileFeliCaRequest.WEBTO;
34     request.url = "http://mobile.sony.jp/";
35     fc.access(request);
36 }
37
38 private function onOpenFailure(evt:OpenStatusEvent):void
39 {
40     var error:Error = evt.object as Error;
41     var msg:String = "NFC Proxy との相互認証に失敗しました。" +
42                     "Error: (" + error.errorID + ") " + error.message;
43     mx.controls.Alert.show(msg);
44 }
45
46 private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
47 {
48 }
49
50 private function onFeliCaAccessFailure(evt:FeliCaStatusEvent):void
51 {
52 }
53
54 private function onFeliCaAccessParameterError(evt:FeliCaStatusEvent):void
55 {
56 }
57
58
59

```

図 4-21 : ブラウザ起動用イベントリスナー登録

ブラウザ起動に成功した場合には、FeliCaStatusEvent クラスの object プロパティに MobileFeliCaResponse が設定されています。

MobileFeliCaResponse クラスに設定されている情報を、表 4-4 に示します。

表 4-4 : MobileFeliCaResponse クラスのプロパティ

No	プロパティ	説明
1	felicaError	FeliCa アクセスライブラリのエラーコードが設定される
2	rwError	リーダー/ライターコントロールライブラリのエラーコードが設定される
3	felicaProxyError	NFCProxyService のエラーコードが設定される
4	data	おサイフケータイのフリー領域の読み出しデータが設定される ※ブラウザ起動処理ではこのプロパティは使用しない

表 4-4 記載の各プロパティをアラート(Alert)で表示するように、onFeliCaAccessCompletek 関数の中に処理を記述します(図 4-22 の黒枠部)。

※ ブラウザ起動に成功した場合の各プロパティの値(エラーコード)は、「0」です。

```

46 private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
47 {
48     var response:MobileFeliCaResponse = evt.object as MobileFeliCaResponse;
49     var msg:String = "ブラウザ起動に成功しました\n" +
50                     "Felica Error Code (" + response.felicaError + ") \n" +
51                     "RW Error Code (" + response.rwError + ") \n" +
52                     "FelicaProxy Error Code (" + response.felicaProxyError + ")";
53     mx.controls.Alert.show(msg);
54
55     fc.close();
56 }

```

図 4-22 : ブラウザ起動成功

ブラウザ起動に失敗した場合のFeliCaStatusEvent クラスの object プロパティには、FeliCaAccessError クラスかError クラスのどちらかが設定されています。

設定されるクラスの違いは、エラー内容によって異なります(表 4-5)。

表 4-5 : エラー情報の違い

No	クラス	説明
1	FeliCaAccessError	FeliCa カードへのアクセスに失敗した場合に設定される ※SDK for FeilCa でエラーが発生した場合に設定される
2	Error	FeliCa カードへのアクセスを実行する際にパラメータ設定エラーなどが発生した場合に設定される

FeliCaAccessError クラスに設定されている情報を、表 4-6 に示します。

表 4-6 : FeliCaAccessError クラス

No	プロパティ	説明
1	errorID	エラーメッセージに関連付けられた参照番号が設定される
2	message	エラーメッセージが設定される
3	feliCaError	FeliCa アクセスライブラリのエラーコードが設定される
4	rwError	リーダ/ライタコントロールライブラリのエラーコードが設定される
5	feliCaProxyError	NFCProxyService のエラーコードが設定される
6	statusFlag1	ステータスフラグ 1 のエラーコードが設定される
7	statusFlag2	ステータスフラグ 2 のエラーコードが設定される

Error クラスの詳細については、ActionScript のヘルプを参照してください。

ブラウザ起動に失敗した場合のエラー情報を、アラート(Alert)で表示するように、onFeliCaAccessFailure 関数の中に処理を記述します(図 4-23 の黒枠部)。

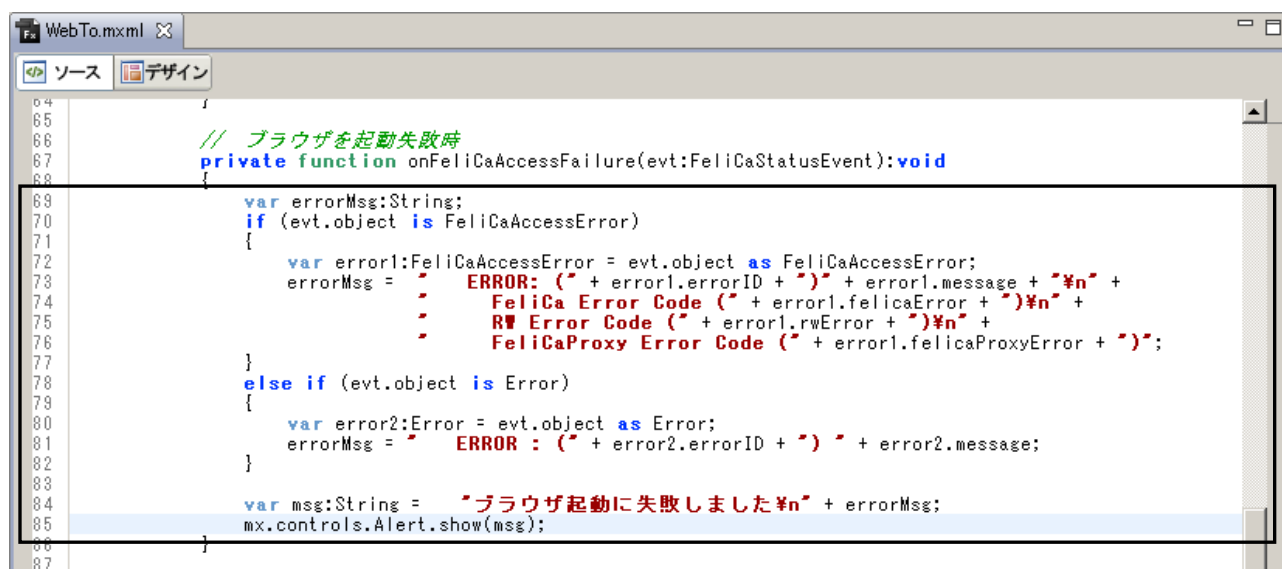


図 4-23 : ブラウザ起動失敗

パラメータ設定エラーの場合には、FeliCaStatusEvent クラスの object プロパティに Error が設定されていますので、同様に onFeliCaAccessFailure 関数の中に処理を記述します(図 4-24 の黒枠部)。

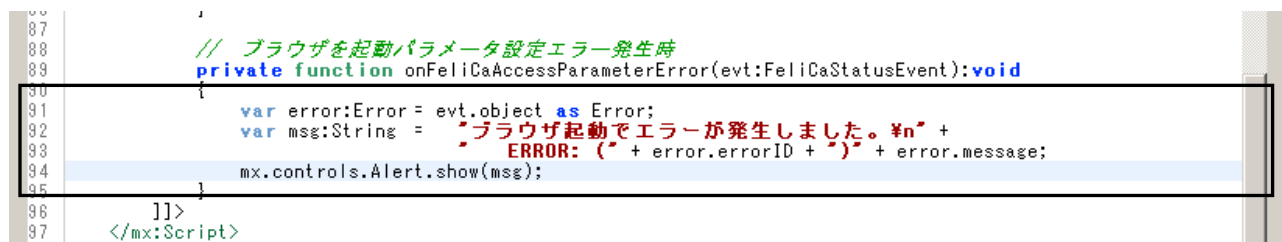


図 4-24 : ブラウザ起動パラメータエラー

■ NFCProxyService の切断

NFCProxyService の切断には、FeliCaControl クラスの close 関数を使用します。

NFCProxyService と認証を行なった場合には、必ず、NFCProxyService の切断を行うようにしてください。

図 4-25 の黒枠部のコードを入力して、保存します。

```
46 private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
47 {
48     var response:MobileFeliCaResponse = evt.object as MobileFeliCaResponse;
49     var msg:String = "ブラウザ起動に成功しました\n" +
50                     "FeliCa Error Code (" + response.felicaError + ") \n" +
51                     "RW Error Code (" + response.rwError + ") \n" +
52                     "FeliCaProxy Error Code (" + response.felicaProxyError + ")";
53     mx.controls.Alert.show(msg);
54 }
55 fc.close();
56 }
57
58 private function onFeliCaAccessFailure(evt:FeliCaStatusEvent):void
59 {
60     var errorMsg:String;
61     if (evt.object is FeliCaAccessError)
62     {
63         var error1:FeliCaAccessError = evt.object as FeliCaAccessError;
64         errorMsg = "ERROR: (" + error1.errorID + ") " + error1.message + "\n" +
65                 "FeliCa Error Code (" + error1.felicaError + ") \n" +
66                 "RW Error Code (" + error1.rwError + ") \n" +
67                 "FeliCaProxy Error Code (" + error1.felicaProxyError + ")";
68     }
69     else if (evt.object is Error)
70     {
71         var error2:Error = evt.object as Error;
72         errorMsg = "ERROR: (" + error2.errorID + ") " + error2.message;
73     }
74
75     var msg:String = "ブラウザ起動に失敗しました\n" + errorMsg;
76     mx.controls.Alert.show(msg);
77 }
78 fc.close();
79 }
80
81 private function onFeliCaAccessParameterError(evt:FeliCaStatusEvent):void
82 {
83     var error:Error = evt.object as Error;
84     var msg:String = "ブラウザ起動でエラーが発生しました。 \n" +
85                     "ERROR: (" + error.errorID + ") " + error.message;
86     mx.controls.Alert.show(msg);
87 }
88 fc.close();
89 }
90 }
91
```

図 4-25 : NFCProxyService の切断

■ アプリケーションの実行

アプリケーションの実行を行う前に、以下のことを確認してください。

- NFCProxyService が開始していること
- リーダ／ライターが接続されていること


作成したアプリケーションの動作確認を行うため、デバッグでアプリケーションを実行します。
ツールボタンのデバッグボタンをクリックします（図 4-26）。



図 4-26 : ツールボタンのデバッグボタンで実行

デバッグボタンをクリックするとブラウザが起動して、作成した Flash の Web ページが表示されます（図 4-27）。

※ Flash アプリケーションをデバッグ起動するためには、Adobe Flash Player デバッグ版が必要です。

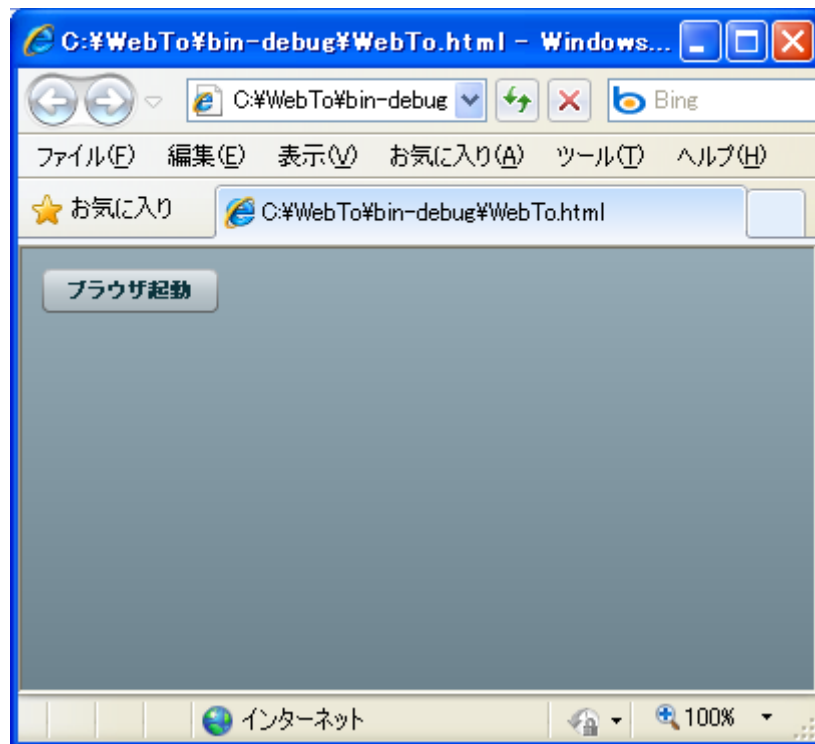


図 4-27 : 作成した Flash を表示

リーダー/ライターにおサイフケータイをかざしたまま、表示されている Flash の「ブラウザ起動」ボタンをクリックすると、ブラウザ起動結果が表示されます（図 4-28）。

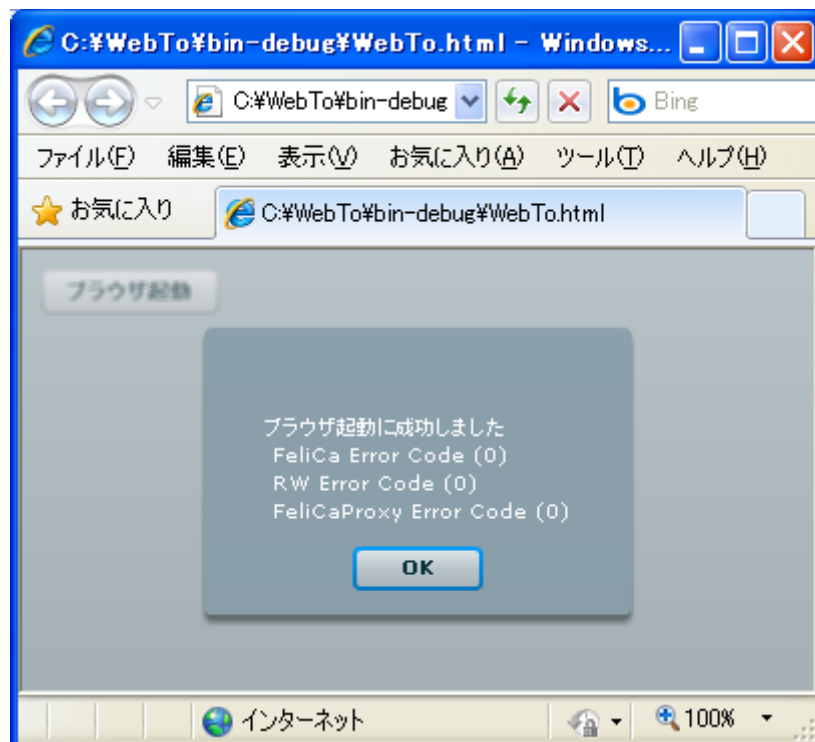
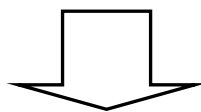
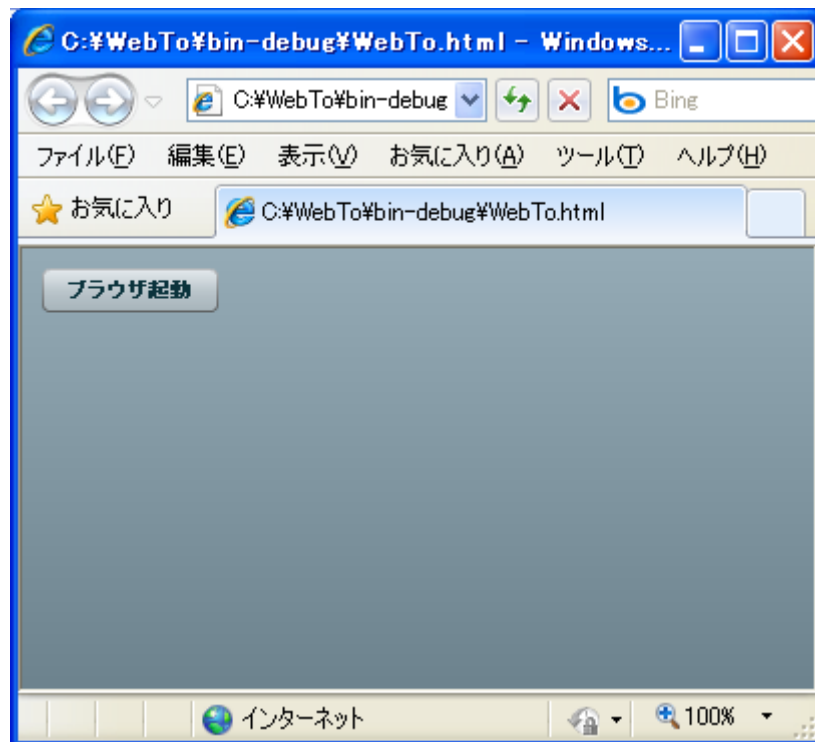


図 4-28 : ブラウザ起動の動作確認

5. SDK for NFC & Adobe® AIR®/Adobe® Flash®の機能を理解する

本章では、SDK for NFC & Adobe® AIR®/Adobe® Flash®が提供する機能について説明します。

本 SDK には Basic 版、Standard 版という 2 種類のパッケージが存在します(「2.1 SDK for NFC & Adobe® AIR® / Adobe® Flash®とは」参照)。

5.1 Basic 版および Standard 版の共通機能

Basic 版および Standard 版共通の機能を、表 5-1 に示します。

表 5-1 : 共通機能

No	機能
1	おサイフケータイのフリー領域*1の読み出し
2	おサイフケータイのフリー領域の書き込み
3	i アプリの起動
4	ブラウザの起動
5	おサイフケータイのフリー領域を読み込んだ後に i アプリの起動
6	おサイフケータイのフリー領域へ書き込んだ後に i アプリの起動
7	NFC フォーラム Type 2、3、4 Tag データの読み出し
8	NFC フォーラム Type 2、3、4 Tag データの書き込み
9	リーダー/ライター占有
10	リーダー/ライターのオープン
11	リーダー/ライターのクローズ
12	FeliCa カード*2の捕捉
13	コンテンツ発行情報*3の取得

*1 株式会社 NTT ドコモのおサイフケータイに設定されている領域です。

*2 FeliCa カードだけではなく、携帯電話などに搭載された FeliCa カード機能も含みます。

Basic 版では特定のシステムコードを持つカードのみ捕捉できます。

*3 モバイル FeliCa OS Version 2.0 以降のモバイル FeliCa IC チップに対するアクセスのみに有効な機能です。

5.2 Standard 版のみの機能

Standard 版のみの機能を、表 5-2 に示します。

表 5-2 : Standard 版のみの機能

No	機能
1	FeliCa アクセスライブラリのバージョン情報の取得
2	FeliCa アクセスライブラリのバージョン番号の取得
3	FeliCa アクセスライブラリの著作権情報の取得
4	FeliCa アクセスライブラリとリーダー/ライターコントロールライブラリのエラー情報の取得
5	FeliCa アクセスライブラリとリーダー/ライターコントロールライブラリのエラー情報リストの取得
6	FeliCa アクセスライブラリとリーダー/ライターコントロールライブラリのエラー情報リストのクリア
7	リーダー/ライターのモード確認
8	リーダー/ライターのオープン状態取得
9	リーダー/ライターの再接続
10	リーダー/ライター種別と接続方式の取得
11	タイムアウト値の設定
12	タイムアウト値の取得
13	リトライカウント値の設定
14	リトライカウント値の取得
15	エリア・サービス鍵バージョンの取得
16	システムコードのリストを取得
17	FeliCa カードの捕捉
18	IDm の取得
19	FeliCa カードのモード取得
20	非セキュリティエリアからのデータの読み出し
21	非セキュリティエリアへのデータの書き込み
22	任意の FeliCa カードコマンドの実行

6. クラスを理解する

本章では、SDK for NFC & Adobe® AIR®/Adobe® Flash®が提供するクラスについて説明します。

Basic 版、Standard 版それぞれのパッケージ製品で用意されているクラス一覧を表示します。

また、「5. SDK for NFC & Adobe® AIR®/Adobe® Flash®の機能を理解する」の機能とクラスの対応表を示しますので、開発の参考にしてください。

各クラスのメソッドやプロパティの詳細については、API Documentation を参照してください。

6.1 クラス一覧

Basic 版および Standard 版共通のクラス一覧を、表 6-1 に示します。

表 6-1 : Basic/Standard 版共通のクラス一覧

パッケージ : com.sony.jp.felica		
No	クラス	概要
1	FeliCaControl	FeliCa アクセスコントロールクラス
2	MobileFeliCaRequest	おサイフケータイへのアクセス要求情報を保持するクラス
3	MobileFeliCaResponse	おサイフケータイへのアクセス結果情報を保持するクラス
4	NFCType2TagRequest	NFC フォーラム Type2 Tag アクセス情報を保持するクラス
5	NFCType2TagResponse	NFC フォーラム Type2 Tag アクセス結果情報を保持するクラス
6	NFCType3TagRequest	NFC フォーラム Type3 Tag アクセス情報を保持するクラス
7	NFCType3TagResponse	NFC フォーラム Type3 Tag アクセス結果情報を保持するクラス
8	NFCType4TagRequest	NFC フォーラム Type 4a、4b Tag アクセス情報を保持するクラス
9	NFCType4TagResponse	NFC フォーラム Type 4a、4b Tag アクセス結果情報を保持するクラス
10	FeliCaSessionRequest	リーダ/ライタを占有するための情報を保持するクラス
11	FeliCaSessionResponse	リーダ/ライタを占有要求した結果情報を保持するクラス
12	FeliCaOpenReaderWriterAutoRequest	リーダ/ライタのオープンを行う情報を保持するクラス
13	FeliCaOpenReaderWriterAutoResponse	リーダ/ライタのオープンを実行した結果情報を保持するクラス
14	FeliCaCloseReaderWriterRequest	リーダ/ライタのクローズを行う情報を保持するクラス
15	FeliCaCloseReaderWriterResponse	リーダ/ライタのクローズを実行した結果情報を保持するクラス
16	FeliCaPollingAndGetCardInformationRequest	Polling コマンドを行う情報を保持するクラス
17	FeliCaPollingAndGetCardInformationResponse	Polling コマンドを実行した結果情報を保持するクラス
18	FeliCaGetContainerIssueInformationRequest	Get Container Issue Information コマンドを行う情報を保持するクラス
19	FeliCaGetContainerIssueInformationResponse	Get Container Issue Information コマンドを実行した結果情報を保持するクラス
パッケージ : com.sony.jp.felica.event		
20	OpenStatusEvent	SDK for AIR での認証結果のイベント発生時にイベントリスナーにパラメータとして渡されます。
21	FeliCaStatusEvent	SDK for AIR での処理結果のイベント発生時にイベントリスナーにパラメータとして渡されます。
パッケージ : com.sony.jp.felica.error		
22	FeliCaAccessError	FeliCa アクセスで発生したエラーに関する情報が含まれています。

Standard 版のみのクラス一覧を、表 6-2 に示します。

表 6-2 : Standard 版のみのクラス一覧

パッケージ : com.sony.jp.felica		
No	クラス	概要
1	FeliCaGetVersionInformationRequest	バージョン情報取得を行う情報を保持するクラス
2	FeliCaGetVersionInformationResponse	バージョン情報取得結果を保持するクラス
3	FeliCaGetVersionNumberRequest	バージョン番号取得を行う情報を保持するクラス
4	FeliCaGetVersionNumberResponse	バージョン番号取得結果を保持するクラス
5	FeliCaGetCopyrightInformationRequest	Copyright 情報取得を行う情報を保持するクラス
6	FeliCaGetCopyrightInformationResponse	Copyright 情報取得結果を保持するクラス
7	FeliCaGetLastErrorTypesRequest	エラー情報取得を行う情報を保持するクラス
8	FeliCaGetLastErrorTypesResponse	エラー情報取得結果を保持するクラス
9	FeliCaGetErrorTypesListRequest	エラー情報リスト取得を行う情報を保持するクラス
10	FeliCaGetErrorTypesListResponse	エラー情報リスト取得結果を保持するクラス
11	FeliCaClearErrorTypesListRequest	エラー情報クリアを行う情報を保持するクラス
12	FeliCaClearErrorTypesListResponse	エラー情報クリア結果を保持するクラス
13	FeliCaReaderWriterIsAliveRequest	リーダ/ライタ生存確認を行う情報を保持するクラス
14	FeliCaReaderWriterIsAliveResponse	リーダ/ライタ生存確認結果を保持するクラス
15	FeliCaReaderWriterIsOpenRequest	リーダ/ライタのオープン状態確認を行う情報を保持するクラス
16	FeliCaReaderWriterIsOpenResponse	リーダ/ライタのオープン状態確認結果を保持するクラス
17	FeliCaReconnectReaderWriterRequest	リーダ/ライタ再接続を行う情報を保持するクラス
18	FeliCaReconnectReaderWriterResponse	リーダ/ライタ再接続結果を保持するクラス
19	FeliCaGetDeviceInformationRequest	リーダ/ライタ情報取得を行う情報を保持するクラス
20	FeliCaGetDeviceInformationResponse	リーダ/ライタ情報取得結果を保持するクラス
21	FeliCaSetTimeoutRequest	タイムアウト値の設定を行う情報を保持するクラス
22	FeliCaSetTimeoutResponse	タイムアウト値の設定結果を保持するクラス
23	FeliCaGetTimeoutRequest	タイムアウト値の取得を行う情報を保持するクラス
24	FeliCaGetTimeoutResponse	タイムアウト値の取得結果を保持するクラス
25	FeliCaSetRetryCountRequest	リトライ回数の設定を行う情報を保持するクラス
26	FeliCaSetRetryCountResponse	リトライ回数の設定結果を保持するクラス
27	FeliCaGetRetryCountRequest	リトライ回数の取得を行う情報を保持するクラス
28	FeliCaGetRetryCountResponse	リトライ回数の取得結果を保持するクラス
29	FeliCaRequestServiceRequest	RequestService を行う情報を保持するクラス
30	FeliCaRequestServiceResponse	RequestService 実行結果を保持するクラス
31	FeliCaRequestSystemCodeRequest	RequestSystemCode を行う情報を保持するクラス
32	FeliCaRequestSystemCodeResponse	RequestSystemCode 実行結果を保持するクラス
33	FeliCaReadBlockWithoutEncryptionRequest	ReadBlockWithoutEncryption を行う情報を保持するクラス
34	FeliCaReadBlockWithoutEncryptionResponse	ReadBlockWithoutEncryption 実行結果を保持するクラス
35	FeliCaWriteBlockWithoutEncryptionRequest	WriteBlockWithoutEncryption を行う情報を保持するクラス
36	FeliCaWriteBlockWithoutEncryptionResponse	WriteBlockWithoutEncryption 実行結果を保持するクラス
37	FeliCaCommunicateThruRequest	任意の FeliCa コマンドの実行を行う情報を保持するクラス
38	FeliCaCommunicateThruResponse	任意の FeliCa コマンドを実行した結果情報を保持するクラス
39	EdyAccess	Edy 履歴読み取りを実行するためのクラス
40	EdyTransaction	EdyEvent 内の Edy 履歴情報を保持するクラス
41	EdyEvent	Edy 履歴読み取り結果をイベントとして通知するためのクラス

6.2 クラスと機能の対応

基本的に1つの機能に対して、対応するクラスは2つあります。

それぞれのクラスの名前は、「xxxRequest」クラス、「xxxResponse」クラスとなっています。

xxxRequest クラスは、各処理(機能)を実行するのに必要な情報を設定するクラスです。

xxxResponse クラスは、各処理(機能)を実行した結果情報を保持しているクラスです。

Basic 版および Standard 版共通の機能とクラスの対応を、表 6-3 に示します。

表 6-3 : Basic/Standard 版共通の機能とクラスの対応表

No	機能	クラス
1	おサイフケータイのフリー領域の読み出し	MobileFeliCaRequest
2	おサイフケータイのフリー領域の書き込み	MobileFeliCaResponse
3	i アプリの起動	
4	ブラウザの起動	
5	おサイフケータイのフリー領域を読み込んだ後に i アプリの起動	
6	おサイフケータイのフリー領域へ書き込んだ後に i アプリの起動	
7	NFC フォーラム Type2 Tag データの読み出し	NFCType2TagRequest
8	NFC フォーラム Type2 Tag データの書き込み	NFCType2TagResponse
9	NFC フォーラム Type3 Tag データの読み出し	NFCType3TagRequest
10	NFC フォーラム Type3 Tag データの書き込み	NFCType3TagResponse
11	NFC フォーラム Type4 Tag データの読み出し	NFCType4TagRequest
12	NFC フォーラム Type4 Tag データの書き込み	NFCType4TagResponse
13	リーダ/ライタ占有	FeliCaSessionRequest FeliCaSessionResponse
14	リーダ/ライタのオープン	FeliCaOpenReaderWriterAutoRequest FeliCaOpenReaderWriterAutoResponse
15	リーダ/ライタのクローズ	FeliCaCloseReaderWriterRequest FeliCaCloseReaderWriterResponse
16	FeliCa カードの捕捉	FeliCaPollingAndGetCardInformationRequest FeliCaPollingAndGetCardInformationResponse
17	コンテナ発行情報の取得	FeliCaGetContainerIssueInformationRequest FeliCaGetContainerIssueInformationResponse

Standard 版のみの機能とクラスの対応表を、表 6-4 に示します。

表 6-4 : Standard 版のみの機能とクラスの対応表

No	機能	クラス
1	FeliCa アクセスライブラリのバージョン情報の取得	FeliCaGetVersionInformationRequest FeliCaGetVersionInformationResponse
2	FeliCa アクセスライブラリのバージョン番号の取得	FeliCaGetVersionNumberRequest FeliCaGetVersionNumberResponse
3	FeliCa アクセスライブラリの著作権情報の取得	FeliCaGetCopyrightInformationRequest FeliCaGetCopyrightInformationResponse
4	FeliCa アクセスライブラリとリーダー/ライタコントロールライブラリのエラー情報の取得	FeliCaGetLastErrorTypesRequest FeliCaGetLastErrorTypesResponse
5	FeliCa アクセスライブラリとリーダー/ライタコントロールライブラリのエラー情報リストの取得	FeliCaGetErrorTypesListRequest FeliCaGetErrorTypesListResponse
6	FeliCa アクセスライブラリとリーダー/ライタコントロールライブラリのエラー情報リストのクリア	FeliCaClearErrorTypesListRequest FeliCaClearErrorTypesListResponse
7	リーダー/ライタのモード確認	FeliCaReaderWriterIsAliveRequest FeliCaReaderWriterIsAliveResponse
8	リーダー/ライタのオープン状態取得	FeliCaReaderWriterIsOpenRequest FeliCaReaderWriterIsOpenResponse
9	リーダー/ライタの再接続	FeliCaReconnectReaderWriterRequest FeliCaReconnectReaderWriterResponse
10	リーダー/ライタ種別と接続方式の取得	FeliCaGetDeviceInformationRequest FeliCaGetDeviceInformationResponse
11	タイムアウト値の設定	FeliCaSetTimeoutRequest FeliCaSetTimeoutResponse
12	タイムアウト値の取得	FeliCaGetTimeoutRequest FeliCaGetTimeoutResponse
13	リトライカウント値の設定	FeliCaSetRetryCountRequest FeliCaSetRetryCountResponse
14	リトライカウント値の取得	FeliCaGetRetryCountRequest FeliCaGetRetryCountResponse
15	エリア・サービス鍵バージョンの取得	FeliCaRequestServiceRequest FeliCaRequestServiceResponse
16	システムコードのリストを取得	FeliCaRequestSystemCodeRequest FeliCaRequestSystemCodeResponse
17	FeliCa カードのモード取得	FeliCaRequestResponseRequest FeliCaRequestResponseResponse
18	非セキュリティエリアからのデータの読み出し	FeliCaReadBlockWithoutEncryptionRequest FeliCaReadBlockWithoutEncryptionResponse
19	非セキュリティエリアへのデータの書き込み	FeliCaWriteBlockWithoutEncryptionRequest FeliCaWriteBlockWithoutEncryptionResponse

6.3 イベントの種類

SDK for NFC & Adobe® AIR®/Adobe® Flash®で定義しているイベントの種類は、表 6-5 の 3 種類です。

表 6-5 : イベントの種類

No	クラス	イベントタイプ定数	説明
1	OpenStatusEvent	OPEN_COMPLETE	NFCProxyService との認証に成功した時に送出される
2		OPEN_FAILURE	NFCProxyService との認証に失敗した時に送出される
3	FeliCaStatusEvent	FELICA_ACCESS_COMPLETE	FeliCa、NFC フォーラム Type2、4 Tag デバイスへのアクセスが成功した時に送出される
4		FELICA_ACCESS_FAILURE	FeliCa、NFC フォーラム Type2、4 Tag デバイスへのアクセス処理でエラーが発生した時に送出される
5		FELICA_ACCESS_PARAMETER_ERROR	パラメータに不備がある時に送出される
6	EdyEvent	SUCCESS	Edy 履歴読み取り成功時に送出される
7		FAILURE	Edy 履歴読み取り失敗時に送出される

OpenStatusEvent クラスおよび FeliCaStatusEvent クラスのイベントタイプは、FeliCaControl クラスの addEventListener 関数で登録します。EdyEvent クラスのイベントタイプは、EdyAccess クラスの addEventListener 関数で登録します。

NFCProxyService との認証時には、OpenStatusEvent.OPEN_COMPLETE と OpenStatusEvent.OPEN_FAILURE を使用します。

おサイフケータイ、NFC フォーラム Tag、FeliCa カードへのアクセス時には、

FeliCaStatusEvent.FELICA_ACCESS_COMPLETE、FeliCaStatusEvent.FELICA_ACCESS_FAILURE、

FeliCaStatusEvent.FELICA_ACCESS_PARAMETER_ERROR を使用します。

Edy 履歴読み取り時には、EdyEvent.SUCCESS、EdyEvent.FAILURE を使用します。

7. クラスを活用する

本章では、複数のクラスを利用して FeliCa カードへアクセスを行うソースコード例を説明します。

NFCProxyService との認証や各パッケージのインポート、イベントリスナーの登録については、「4. アプリケーションの開発」を参照してください。

7.1 おサイフケータイにアクセスする

おサイフケータイにアクセスするアプリケーションを作成します。

アプリケーションの処理の流れは、以下のとおりとします。

【処理の流れ】

1. リーダ／ライタを占有する
2. フリー領域を読み込む
3. ブラウザ起動を行う
4. リーダ／ライタ占有を開放する

以下に必要なインポートパッケージ（表 7-1）と処理ごとのソースコードを示します。

※サンプルプロジェクトあり（プロジェクト名：MobileAccess）

表 7-1：インポートパッケージ一覧

No	インポートパッケージ
1	com.sony.jp.felica.FeliCaSessionRequest
2	com.sony.jp.felica.FeliCaSessionResponse
3	com.sony.jp.felica.MobileFeliCaRequest
4	com.sony.jp.felica.MobileFeliCaResponse

■ リーダ／ライタを占有する

```
// リーダ／ライタの占有
private function RwLock():void
{
    // リーダ／ライタを占有する為の情報の設定
    var request:FeliCaSessionRequest = new FeliCaSessionRequest();

    request.type          = FeliCaSessionRequest.HOLD; // リーダ／ライタ占有要求
    request.lockTimeout   = 10; // 占有(Lock)するまでのタイムアウト時間(秒)
    request.unlockTimeout = 60; // 占有(Lock)してから開放(Unlock)するまでの見込み時間(秒)
                                // ※ NFCProxyはこのunlockTimeout時間経過した場合
                                // 自動でリーダ／ライタ開放を行う

    // リーダ／ライタ占有を実行
    fc.access(request);
}
```

```
// FeliCa アクセス成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaSessionResponse)
    {
        var session:FeliCaSessionResponse = evt.object as FeliCaSessionResponse;

        if (session.type == FeliCaSessionResponse.HOLD)
        {
            // リーダ／ライタの占有成功時の処理を記述する
            // ここでは、次の処理にあたるフリー領域の読み込み処理を行う
            MobileRead(); // この関数については「フリー領域を読み込む」を参照
        }
    }
}
```

■ フリー領域を読み込む

フリー領域を読み込むには、事前におサイフケータイに i アプリを登録している必要があります。

```
// フリー領域を読み込む
private function MobileRead():void
{
    // 携帯電話のフリー領域を読み込みする為の情報の設定
    var request:MobileFeliCaRequest = new MobileFeliCaRequest();

    request.type = MobileFeliCaRequest.READ; // おサイフケータイのフリー領域読み込み要求
    request.cpid = "AD4D5DB9B91C41DC933E79C81AAFDB3B"; // フリーエリアを特定するための CPID
    request.pin = "12345678"; // PIN 情報

    // フリー領域の読み込みを実行
    fc.access(request);
}
```

```
// FeliCa アクセス成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaSessionResponse)
    {
        // 略
    }
    if (evt.object is MobileFeliCaResponse)
    {
        var response:MobileFeliCaResponse = evt.object as MobileFeliCaResponse;

        if (response.type == MobileFeliCaResponse.READ)
        {
            // フリー領域のデータ読み込み成功時の処理を記述する
            // ここでは、次の処理にあたるフリー領域の読み込み処理を行う

            // ※ 読み込んだ [data] プロパティに設定されている
            trace("読み込みデータ");
            trace(response.data);

            WebTo(); // この関数については「ブラウザ起動を行う」を参照
        }
    }
}
```

■ ブラウザ起動を行う

ブラウザ起動機能は、おサイフケータイであれば、どの機種でも行うことが可能です。

```
// ブラウザ起動を実行
private function WebTo():void
{
    // 携帯電話のブラウザを起動する為の情報の設定
    var request:MobileFeliCaRequest = new MobileFeliCaRequest();

    request.type = MobileFeliCaRequest.WEBTO; // ブラウザ起動要求
    request.url = "http://mobile.sony.jp/"; // ブラウザ起動時の URL

    // 携帯電話のブラウザを起動
    fc.access(request);
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaSessionResponse)
    {
        // 略
    }
    if (evt.object is MobileFeliCaResponse)
    {
        var response:MobileFeliCaResponse = evt.object as MobileFeliCaResponse;

        if (response.type == MobileFeliCaResponse.READ) {
            // 略
        }
        if (response.type == MobileFeliCaResponse.WEBTO)
        {
            // ブラウザ起動成功時の処理を記述する
            // ここでは、次の処理にあたるリーダ／ライタ占有開放処理を行う
            RwUnlock(); // この関数については「リーダ／ライタ占有を開放する」を参照
        }
    }
}
```

■ リーダ／ライター占有を開放する

```
// リーダ／ライター占有の開放を実行
private function RwUnlock():void
{
    // リーダ／ライター占有を開放する為の情報の設定
    var request:FeliCaSessionRequest = new FeliCaSessionRequest();

    request.type = FeliCaSessionRequest.RELEASE;    // リーダ／ライター占有を開放要求

    // リーダ／ライター占有解放を実行
    fc.access(request);
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaSessionResponse)
    {
        var session:FeliCaSessionResponse = evt.object as FeliCaSessionResponse;

        if (session.type == FeliCaSessionResponse.HOLD)
        {
            // 略
        }
        if (session.type == FeliCaSessionResponse.RELEASE)
        {
            // リーダ／ライターの占有の開放成功時の処理を記述する
            // ここでは、次の処理にあたる NFCProxy との切断処理を行う
            fc.close();
        }
    }
    if (evt.object is MobileFeliCaResponse) {
        // 略
    }
}
```


7.2 NFC フォーラム Tag にアクセスする

NFC フォーラム Tag にアクセスするアプリケーションを作成します。

アプリケーションの処理の流れは、以下のとおりとします。

本節では、NFC フォーラム Type3 Tag について記述しますが、Type2、Type4 の場合は NFCType3TagRequest (Response) を NFCType2TagRequest (Response)、NFCType4TagRequest (Response) に書き替えてください。

【処理の流れ】

1. NFC フォーラム Type3 Tag のデータを読み込む
2. NFC フォーラム Type3 Tag のデータを書き込む

以下に必要なインポートパッケージ（表 7-2）と処理ごとのソースコードを示します。

※サンプルプロジェクトあり（プロジェクト名：NFCType3TagAccess）

表 7-2：インポートパッケージ一覧

No	インポートパッケージ
1	com.sony.jp.felica.NFCType3TagRequest
2	com.sony.jp.felica.NFCType3TagResponse

■ NFC フォーラム Type3 Tag のデータを読み込む

```
// NFC フォーラム Type3 Tag の読み込みを実行
private function NFCType3TagRead():void
{
    // NFC フォーラム Type3 Tag のデータを読み込みする為の情報の設定
    var request:NFCType3TagRequest = new NFCType3TagRequest();

    request.type = NFCType3TagRequest.READ; // NFC フォーラム Type3 Tag データを読み込み要求

    // NFC フォーラム Type3 Tag データの読み込みを実行
    fc.access(request);
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaSessionResponse)
    {
        // 略
    }
    if (evt.object is NFCType3TagResponse)
    {
        var response:NFCType3TagResponse = evt.object as NFCType3TagResponse;

        if (response.type == NFCType3TagResponse.READ)
        {
            // NFC フォーラム Type3 Tag のデータ読み込み成功時の処理を記述する
            // ここでは、次の処理にあたる NFC フォーラム Type3 Tag の書き込み処理を行う

            // ※ 読み込んだ [data] プロパティに設定されている
            trace("読み込みデータ : " + response.data);

            NFCType3TagWrite(); // この関数については「NFC フォーラム Type3 Tag のデータを書
                               // き込む」を参照
        }
    }
}
```

■ NFC フォーラム Type3 Tag のデータを書き込む

```
// NFC フォーラム Type3 Tag の書き込みを実行
private function NFCType3TagWrite():void
{
    // NFC フォーラム Type3 Tag のデータを書き込みする為の情報の設定
    var request:NFCType3TagRequest = new NFCType3TagRequest();

    request.type = NFCType3TagRequest.WRITE; // NFC フォーラム Type3 Tag データを書き込み要求

    // 書き込むデータは 16 進文字列で指定します。
    // 最大数は 16 バイト * 13 ブロックです。

    // 書き込みたい文字列を byte 配列にする
    var data:ByteArray = new ByteArray();
    data.writeUTFBytes("Sample Application ....");

    // byte 型配列を 16 進数表記の文字列に変換して data プロパティに設定
    request.data = bytesToString(data);

    // NFC フォーラム Type3 Tag データの書き込みを実行
    fc.access(request);
}

// byte 型配列を 16 進数表記の文字列に変換
private function bytesToString(data:ByteArray):String
{
    var result:String = "";
    var temp:String;

    for (var i:uint = 0; i < data.length; i++)
    {
        temp = data[i].toString(16);
        if (temp.length != 2)
        {
            temp = "0" + temp;
        }
        result += temp;
    }
    return result;
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is NFCType3TagResponse)
    {
        var response:NFCType3TagResponse = evt.object as NFCType3TagResponse;

        if (response.type == NFCType3TagResponse.READ)
        {
            // 略
        }
        if (response.type == NFCType3TagResponse.WRITE)
        {
            // 略
        }
    }
}
```

7.3 FeliCa カードを検出する

FeliCa カードを検出するアプリケーションを作成します。

アプリケーションの処理の流れは、以下のとおりとします。

【処理の流れ】

1. リーダ／ライタをオープンする
2. Polling を実行する
3. リーダ／ライタをクローズする

以下に必要なインポートパッケージ（表 7-3）と処理ごとのソースコードを示します。

※サンプルプロジェクトあり（プロジェクト名：FeliCaSearch）

表 7-3：インポートパッケージ一覧

No	インポートパッケージ
1	com.sony.jp.felica.FeliCaOpenReaderWriterAutoRequest
2	com.sony.jp.felica.FeliCaOpenReaderWriterAutoResponse
3	com.sony.jp.felica.FeliCaPollingAndGetCardInformationRequest
4	com.sony.jp.felica.FeliCaPollingAndGetCardInformationResponse
5	com.sony.jp.felica.FeliCaCloseReaderWriterRequest
6	com.sony.jp.felica.FeliCaCloseReaderWriterResponse

■ リーダ／ライタをオープンする

```
// リーダ／ライタのオープンを実行
private function RwOpen():void
{
    // [open_reader_writer_auto] コマンドを実行
    fc.access(new FeliCaOpenReaderWriterAutoRequest());
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaOpenReaderWriterAutoResponse)
    {
        // リーダ／ライタ オープン成功時の処理を記述する
        // ここでは、次の処理にあたる Polling 処理を行う

        Polling(); // この関数については「Polling を実行する」を参照
    }
}
```

■ Polling を実行する

どの FeliCa カードでも検出できるようにシステムコードを FFFFh と設定する。

```
// FeliCa カードを検出するために、Polling の実行
private function Polling():void
{
    // [polling_and_get_card_information] コマンドを実行する為の情報の設定
    var request:FeliCaPollingAndGetCardInformationRequest =
        new FeliCaPollingAndGetCardInformationRequest();

    request.systemCode = "FFFF";    // ポーリングするシステムコード

    // [polling_and_get_card_information] コマンドを実行
    fc.access(request);
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaOpenReaderWriterAutoResponse)
    {
        // 略
    }
    if (evt.object is FeliCaPollingAndGetCardInformationResponse)
    {
        var response:FeliCaPollingAndGetCardInformationResponse =
            evt.object as FeliCaPollingAndGetCardInformationResponse;

        // polling_and_get_card_information 成功時の処理を記述する
        // ここでは、次の処理にあたるリーダ／ライタクローズ処理を行う

        // ※ 検出した FeliCa の情報 (IDm, PMm) はそれぞれ、
        //      [idm][pmm] プロパティに設定されている
        trace("IDm : " + response.idm);
        trace("PMm : " + response.pmm);

        RwClose();    // この関数については「リーダ／ライタをクローズする」を参照
    }
}
```

■ リーダ／ライタをクローズする

```
// リーダ／ライタのクローズを実行
private function RwClose():void
{
    // [close_reader_writer] コマンドを実行
    fc.access(new FeliCaCloseReaderWriterRequest());
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaOpenReaderWriterAutoResponse)
    {
        // 略
    }
    if (evt.object is FeliCaPollingAndGetCardInformationResponse)
    {
        // 略
    }
    if (evt.object is FeliCaCloseReaderWriterResponse)
    {
        // リーダ／ライタクローズ成功時の処理を記述する
        fc.close();
    }
}
```


7.4 連続して FeliCa カードを検出する

一定時間連続して FeliCa カードを検出するアプリケーションを作成します。

アプリケーションの処理の流れは、以下のとおりとします。

【処理の流れ】

1. リーダ／ライタを占有する
2. リーダ／ライタをオープンする
3. Polling を実行する ※ 一定時間連続で実行する
4. リーダ／ライタをクローズする
5. リーダ／ライタ占有を開放する

以下に必要なインポートパッケージ（表 7-4）と処理ごとのソースコードを示します。

※サンプルプロジェクトあり（プロジェクト名：AutoFeliCaSearch）

表 7-4：インポートパッケージ一覧

No	インポートパッケージ
1	com.sony.jp.felica.FeliCaSessionRequest
2	com.sony.jp.felica.FeliCaSessionResponse
3	com.sony.jp.felica.FeliCaOpenReaderWriterAutoRequest
4	com.sony.jp.felica.FeliCaOpenReaderWriterAutoResponse
5	com.sony.jp.felica.FeliCaPollingAndGetCardInformationRequest
6	com.sony.jp.felica.FeliCaPollingAndGetCardInformationResponse
7	com.sony.jp.felica.FeliCaCloseReaderWriterRequest
8	com.sony.jp.felica.FeliCaCloseReaderWriterResponse

基本的な処理は、「7.3 FeliCa カードを検出する」と同じになります。ここでは、「7.3 FeliCa カードを検出する」のソースコード例をそのまま活用し、一定時間連続して Polling を実行するソースコードのみを示します。

■ 一定間隔で Polling を実行する（Polling の連続実行）

Polling を連続実行する場合には、Polling の実行結果を取得後、一定時間待ち合わせてから次の Polling を実行します。

処理を一定間隔待ち合わせるには、Flex ライブラリ内に実装されている `flash.utils` パッケージの `setInterval` 関数および `clearInterval` 関数を使用します。

`setInterval` 関数の戻り値である呼び出し ID は、`clearInterval` 関数の引数になりますので、変数を宣言しておきます。

```
// 連続 Polling を行う間隔に対する一意の数値識別子(呼び出し ID)  
private var waitIntervalId:uint = 0;
```

FeliCa カードの検出結果に関わらず、一定時間待ち合わせてから次の Polling を実行することになります。

「7.3 FeliCa カードを検出する」のソースコードでは、FeliCa カードを検出できた場合に、「リーダー/ライタをクローズする」処理が記述されていますが、その処理を削除し、次のソースコードに修正します。

```
// 成功時  
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void  
{  
    if (evt.object is FeliCaPollingAndGetCardInformationResponse)  
    {  
        // 次の実行処理を行う  
        this.nextProcess();  
    }  
}
```

```
// 次の実行処理を行う  
private function nextProcess():void  
{  
    // 次の Polling まで 500 ミリ秒待ち合わせてから、  
    // タイマー呼び出し用 execPolling 関数を実行する  
    this.waitIntervalId = setInterval(execPolling, 500);  
}
```

FeliCa カードへのアクセスでエラーが発生した場合の処理については、FeliCa カードが検出できなかった場合のみ一定時間待ち合わせてから、次の Polling を実行することになります。

FeliCaAccessError クラスの `rwError` プロパティの値で FeliCa カードが検出できなかったのか、その他のエラーが発生したのかを判断することができます。

FeliCa カードが検出できない場合には、`rwError` プロパティの値が 157 となるため、次のようにソースコードを修正します。

```
// 失敗時
private function onFeliCaAccessFailure(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaAccessError)
    {
        var error1:FeliCaAccessError = evt.object as FeliCaAccessError;

        if (error1.rwError == 157)    // FeliCa が検出できなかった場合
        {
            // 次の実行処理を行う
            this.nextProcess();
        }
        else    // その他のエラーが発生した場合
        {
            // その他のエラーが発生した場合の処理を記述する
        }
    }
}
```

一定時間待ち合わせた時に呼ばれる `execPolling` 関数を定義し、Polling を実行する処理を次のように記述します。

```
// 連続 Polling 実行待ち合わせ完了時
private function execPolling():void
{
    // 指定した setInterval() 呼び出しをキャンセル
    clearInterval(this.waitIntervalId);

    // FeliCa カードを検出するために、Polling の実行
    Polling();
}
```

■ 一定時間経過したら連続 Polling を終了する

一定の間隔で実行している Polling を終了させる必要があるため、一定時間経過したら Polling の連続実行を終了させるようにします。

一定時間待ち合わせてから次の Polling を実行する時と同じように、setInterval 関数および clearInterval 関数を使用しますので、戻り値の呼び出し ID を格納する変数と連続 Polling の終了フラグも宣言します。

```
// 連続 Polling タイムアウトに対する一意の数値識別子(呼び出し ID)
private var pollingIntervalId:uint = 0;

// 連続 Polling 終了フラグ
private var exitPollingFlag:Boolean = false;
```

リーダ／ライタのオープンに成功し、最初の Polling を実行する前に、setInterval 関数を追記して、exitPolling 関数を次のように記述します。

```
// 成功時
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaOpenReaderWriterAutoResponse)
    {
        // 連続 Polling を終了させる関数をミリ秒単位で指定した間隔ごとに実行
        // ※ 今回は 30 秒とする
        this.pollingIntervalId = setInterval(exitPolling, 30000);

        // FeliCa カードを検出するために、Polling の実行
        Polling();
    }
}
```

```
// 連続 Polling 実行終了
private function exitPolling():void
{
    // 連続 Polling 終了フラグのセット
    this.exitPollingFlag = true;

    // 指定した setInterval() 呼び出しをキャンセル
    clearInterval(this.pollingIntervalId);
}
```

注意事項：

連続 Polling の実行時間については、最大で 120 秒、Polling の間隔は 500 ミリ秒以上を目安にアプリケーションの作成をしてください。

「一定間隔で Polling を実行する（Polling の連続実行）」で記述した nextProcess 関数を、次のように修正します。

```
// 次の実行処理を行う
//     連続 Polling をそのまま行うか
//     連続 Polling を終了してリーダー/ライタのクローズを行う
private function nextProcess():void
{
    if (this.exitPollingFlag == true) // 30 秒のタイムアウトが発生していた場合
    {
        // リーダー/ライタのクローズを実行
        RwClose();
    }
    else
    {
        // 次の Polling まで 500 ミリ秒待ち合わせる
        this.waitIntervalId = setInterval(execPolling, 500);
    }
}
```

7.5 FeliCa カードのデータを読み書きする

FeliCa カードのデータを読み書きするアプリケーションを作成します。

アプリケーションの処理の流れは、以下のとおりとします。

【処理の流れ】

1. Standard 版で NFCProxyService と認証する
2. リーダ／ライタを占有する
3. リーダ／ライタをオープンする
4. Polling を実行する
5. FeliCa カードのデータを読み込む
6. FeliCa カードのデータを書き込む
7. リーダ／ライタをクローズする
8. リーダ／ライタ占有を開放する

以下に必要なインポートパッケージ（表 7-5）と処理ごとのソースコードを示します。

※サンプルプロジェクトあり（プロジェクト名：FeliCaReadWrite）

表 7-5：インポートパッケージ一覧

No	インポートパッケージ
1	com.sony.jp.felica.FeliCaSessionRequest
2	com.sony.jp.felica.FeliCaSessionResponse
3	com.sony.jp.felica.FeliCaOpenReaderWriterAutoRequest
4	com.sony.jp.felica.FeliCaOpenReaderWriterAutoResponse
5	com.sony.jp.felica.FeliCaPollingAndGetCardInformationRequest
6	com.sony.jp.felica.FeliCaPollingAndGetCardInformationResponse
7	com.sony.jp.felica.FeliCaReadBlockWithoutEncryptionRequest
8	com.sony.jp.felica.FeliCaReadBlockWithoutEncryptionResponse
9	com.sony.jp.felica.FeliCaWriteBlockWithoutEncryptionRequest
10	com.sony.jp.felica.FeliCaWriteBlockWithoutEncryptionResponse
11	com.sony.jp.felica.FeliCaCloseReaderWriterRequest
12	com.sony.jp.felica.FeliCaCloseReaderWriterResponse

このアプリケーションは Standard 版のみで動作します。

Standard 版の場合、NFCProxyService との認証にはライセンスキーが必要です。

Standard 版での NFCProxyService との認証方法については、以下の「Standard 版で NFCProxyService と認証する」を参照してください。

■ Standard 版で NFCProxyService と認証する

Standard 版で NFCProxyService と認証する場合には、NFCProxy の TCP/IP 通信用ポート番号以外に、以下のデータを指定する必要があります。

- ライセンスキー

Standard 版を購入した際に発行されるライセンスキーを指定します。

- Flash アプリケーションが動作する URL

Flash アプリケーションが動作する URL を指定します。空の文字列を指定することも可能です。

URL を指定した場合は、該当 URL 以外で Flash を配置した場合の動作をブロックします。

特定の URL のみで動作をさせたい場合は、指定してください。また、空の場合はどんな URL でも動作します。

```
// ライセンスキー
var license_key:String = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

// flash アプリケーションが動作する URL
// ※ flash アプリケーションが不正にコピーされたくない場合には、
//     flash が動作する URL を必ず指定する
var location:String = "";

// NFCProxy の認証
fc.open(10250, license_key, location);
```

■ リーダ／ライタを占有する

「7.3 FeliCa カードを検出する」での「リーダー／ライタを占有する」のソースコード例がそのまま活用できます。

■ リーダ／ライタをオープンする

「7.3 FeliCa カードを検出する」での「リーダー／ライタをオープンする」のソースコード例がそのまま活用できます。

■ Polling を実行する

```
// FeliCa カードを検出するために、Polling の実行
private function Polling():void
{
    // Polling コマンドを実行する為の情報の設定
    var request:FeliCaPollingAndGetCardInformationRequest
        = new FeliCaPollingAndGetCardInformationRequest ();

    request.systemCode    = "FFFF";    // ポーリングするシステムコード

    // Polling コマンドを実行
    fc.access(request);
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaPollingAndGetCardInformationResponse)
    {
        var response: FeliCaPollingAndGetCardInformationResponse
            = evt.object as FeliCaPollingAndGetCardInformationResponse;

        // Polling 成功時の処理を記述する
        // ここでは、次の処理にあたる FeliCa カードのデータを読み込む処理を行う

        // ※ 検出した FeliCa の情報 (IDm, PMm) はそれぞれ、
        //      [idm][pmm] プロパティに設定されている
        trace("IDm : " + response.idm);
        trace("PMm : " + response.pmm);

        Read(); // この関数については「FeliCa カードのデータを読み込む」を参照
    }
}
```


■ FeliCa カードのデータを読み出す

今回のアプリケーションでは、Sample5 カード*のデータを読み出すこととします。
読み出すサービスコード、ブロック数などの値は、表 7-6 のとおりです。

* 別売の開発用カード(ICS-E003、ICS-E006、ICS-E007 など)。

表 7-6 : データ読み出し情報

No	名称	値	備考
1	エリア／サービス数	1	－
2	エリア／サービスコードリスト	1009	2 バイトコード指定
3	ブロック数	8	－
4	ブロックリスト	80008001800280038004800580068007	2 バイトエレメント指定

```
// FeliCa カードのデータの読み出し
private function Read():void
{
    // [read_without_encryption] コマンドを実行する為の情報の設定
    var request:FeliCaReadBlockWithoutEncryptionRequest =
        new FeliCaReadBlockWithoutEncryptionRequest();

    request.serviceCodeList.push("1009"); // サービスコードリスト

    for (var i:uint=0; i < 8; i++)
    {
        var strBlockListLower:String = i.toString();
        while(strBlockListLower.length != 2)
        {
            strBlockListLower = "0" + strBlockListLower;
        }
        request.blockList.push("80" + strBlockListLower); // ブロックリスト
    }

    // [read_without_encryption] コマンドを実行
    fc.access(request);
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaReadBlockWithoutEncryptionResponse)
    {
        var response:FeliCaReadBlockWithoutEncryptionResponse =
            evt.object as FeliCaReadBlockWithoutEncryptionResponse;

        // read_without_encryption 成功時の処理を記述する
        // ここでは、次の処理にあたる FeliCa カードのデータを書き込む処理を行う

        // ※ データは[blockData]に格納されている
        this.bufData = "";
        for (var i:uint=0; i < response.blockData.length; i++)
        {
            // 読み出したデータをそのまま書き込みデータとするためにデータを保持する
            this.data += response.blockData[i];
            trace("(" + i.toString() + ") : " + response.blockData[i]);
        }

        Write(); // この関数については「FeliCa カードのデータを書き込む」を参照
    }
}
```

■ FeliCa カードのデータを書き込む

読み出した FeliCa カードのデータをそのまま書き込みます（表 7-6）。

```
// FeliCa カードのデータを書き込み
private function Write():void
{
    var request:FeliCaWriteBlockWithoutEncryptionRequest =
        new FeliCaWriteBlockWithoutEncryptionRequest();

    request.serviceCodeList.push("1009");           // サービスコードリスト

    for (var i:uint=0; i < 8; i++)
    {
        var strBlockListLower:String = i.toString();
        while(strBlockListLower.length != 2)
        {
            strBlockListLower = "0" + strBlockListLower;
        }
        request.blockList.push("80" + strBlockListLower); // ブロックリスト
    }

    request.blockData      = this.bufData;          // 書き込みブロックデータ

    // [write_without_encryption] コマンドを実行
    fc.access(request);
}
```

```
// 成功時 (イベントハンドラ)
private function onFeliCaAccessComplete(evt:FeliCaStatusEvent):void
{
    if (evt.object is FeliCaWriteBlockWithoutEncryptionResponse)
    {
        // write_without_encryption 成功時の処理を記述する
        // ここでは、次の処理にあたるリーダ／ライタクローズ処理を行う
        RwClose(); // この関数については「リーダ／ライタをクローズする」を参照
    }
}
```

■ リーダ／ライタをクローズする

「7.3 FeliCa カードを検出する」での「リーダ／ライタをクローズする」のソースコード例がそのまま活用できます。

■ リーダ／ライタ占有を開放する

「7.1 おサイフケータイにアクセスする」での「リーダ／ライタ占有を開放する」のソースコード例がそのまま活用できます。

8. アプリケーション開発のためのリソース

本文書で説明する SDK の最も基本的な使用方法を超えた、より実用的なアプリケーション開発を行うために、様々なリソースが利用可能です。

8.1 SDK for NFC and Adobe AIR / Adobe Flash Basic API Documentation (ASDoc)

本製品に含まれている SDK for NFC and Adobe AIR / Adobe Flash Basic API Documentation (以下、API Documentation)は、SDK の全ての公開クラスの仕様を記述したものです。アプリケーションを開発する上で、使用するクラスについて参照できます。本文書で紹介したクラスについてさらに詳細に記載されており、また本文書で説明していないクラスの仕様についても記述しています。

API Documentation は 本製品のインストールフォルダ内の/doc/asdoc/index.html に収録しています。

8.2 サンプルプログラム

本製品の/sample フォルダ以下にいくつかのサンプルプログラムのソースファイルを収録しています。詳細については、各サンプルプログラムのフォルダ内の readme ファイルを参照してください。

また、後述する「FeliCa Developers' Blog」からも、いくつかのサンプルプログラムをダウンロードすることができます。

8.3 技術情報（ソニー公式サイト）

- <http://www.sony.co.jp/Products/felica/business/tech-support/index.html>

FeliCa に関する様々な技術情報およびガイドラインを上記のサイトで公開しております。

FeliCa を利用するアプリケーションを開発するにあたり、以下のドキュメントが有用です。

- FeliCa カード ユーザーズマニュアル
- FeliCa カードに関するソフトウェア開発テクニカルノート
- FeliCa カード コマンドシーケンス設計ガイドライン

FeliCa 技術方式を用いた非接触 IC カードを用いてアプリケーション開発を行う際には、カードごとの仕様差異およびセキュリティに十分注意してください。

8.4 NFC フォーラム

- <http://www.nfc-forum.org>

NFC フォーラムは、近距離無線通信の国際標準である NFC 技術の実装と標準化を目的とする団体です。NFC フォーラムは自身の公式サイトで標準化された技術情報を公開しています。本製品の NFC フォーラム Tag へアクセスする機能は NFC フォーラムで公開されている技術仕様に基づいて実装されています。

上記サイトで公開されている技術情報を参照することで、NFC に準拠した機器との連携を行うアプリケーションが開発可能です。例えば、” Smart Poster Record Type Definition” に基づいて URL が書き込まれた NFC フォーラム Type3 Tag は、同仕様に基づいた様々なデバイスおよびアプリケーションで読み取り、その URL を表示させることができます。

8.5 FeliCa Developers' Blog

- http://blog.felicalauncher.com/sdk_for_air/

FeliCa Developers' Blog は、本製品に関する最新の情報を発信しています。

ソフトウェア

SDK for NFC & Adobe® AIR®/Adobe® Flash®

開発手引書 Version 1.3

2009 年 11 月	初版発行	FeliCa デバイス事業部
2011 年 11 月	改訂	FeliCa 事業部

ソニー株式会社

No. M628-J01-30

© 2009,2010,2011 Sony Corporation

Printed in Japan