LemoNovel AS3
ゲーム制作入門

著：アライコウ

『はじめに』

アマチュアでもお手軽にゲームが作れる……そんな便利な時代になって十数年が経とうとしています。

パソコンの発達と普及にともなって次々と生まれてきたゲーム制作ツールは、アマチュアゲーム業界のみならず、プロの業界の発展にも大きく寄与してきました。当初はアマチュアゲームだったものがプロの注目を浴びて商業化を果たしたという例は、枚挙に暇がありません。

しかしこれまでは、それらのツールで制作したゲームは、Windows パソコンでのみ動作するものが大半でした。もし他の環境でも動くゲームを作ろうと思ったら、ツールには頼らず、難しいプログラミング言語を一から覚えなければならないことが多かったのです。

環境の垣根を越えて動作する、クロスプラットフォームの制作ツールがあればいいのにと思ったことはないでしょうか？　今や Mac、Linux、iOS、Android など、Windows 以外のパソコンだけでなく、モバイル環境でもゲームをプレイするユーザーが増えており、こうした状況に対応できる制作ツールは、多くの人が待ち望んでいたものです。

LemoNovel AS3 はクロスプラットフォームで動作する、ノベル・アドベンチャーゲームを主眼とした制作ツールです。また、既存のツールと何ら遜色ない多機能さと応用力を備えており、商業レベルにも耐えうるクオリティの作品を作ることも可能なのです。

その多機能さのため、一冊ですべての機能を書くことは叶わないほどなのですが、本書では入門編として、基本的な使い方を紹介していきます。本書の内容を押さえておけば、充分に遊べる作品を作れますし、さらにステップアップしたい人の一助になることでしょう。

今までゲーム制作をしたことがない人も、他のツールで経験がある人も、ぜひとも LemoNovel AS3 を試してみて、より充実したクリエイティブな日々を送っていただければと思います。

『目次』

第１章

LemoNovel AS3 とは？

『あらゆる環境で作品を制作・リリース』

●Windows 以外でも動作する

LemoNovel AS3 のメリットは、なんといっても環境を問わず作品を制作し、リリースできることです。

システムのベースとなっているのは Adobe 社の Flash または AIR（Adobe Integrated Runtime）で、これらは元来、クロスプラットフォームを目的として開発されています。したがってそのシステムを利用した LemoNovel AS3 も、クロスプラットフォームが実現しているわけです。

さて、LemoNovel AS3 はツール名であると同時に、ノベルシステムそのものを指す名称でもあります。リリースを想定している環境によって、以下の 3 つのいずれかを選択するのです。

・LemoNovel AS3……ウェブブラウザ
・LemoNovel AIR……ローカル環境（ハードディスクにインストール）
・LemoNovel AIR Mobile……iOS（iPhone、iPod touch、iPad）、Android

本書では今後、これらをまとめて指す場合は『LemoNovel AS3』と表記し、個別のシステムを指す場合は『AS3 版』『AIR 版』『AIR Mobile 版』と表記します。これらはすべて同一の素材ファイルで動作するので、たとえばパソコン版からモバイル版の移植も非常に容易になります。

AS3 版が Flash Player、AIR 版と AIR Mobile 版が Adobe AIR を利用します。Flash Player は 11.4 以上、Adobe AIR は 3.5 以上のバージョンが必須となります。これらが快適に動作するなら、あらゆるパソコンやスマートフォンで実行できます。

ただし Linux については、Adobe AIR のバージョンが 2.6 で止まっているため、AIR 版と AIR Mobile 版での制作とリリースはできません。

●どんなゲームを作れるか

LemoNovel AS3 は主としてノベル・アドベンチャーゲームのための制作ツールです。利用にあたっては開発者サイトに掲載されている利用規約に同意する必要がありますが、この規約は非常に緩やかなもので、制作内容についても一切の制限が設けてありません。パソコンゲームで主流の 18 禁美少女ゲームや、二次創作も問題なく作ることができます。

特に AIR Mobile 版で作れる iOS＆Android アプリは、ここ数年で爆発的に流行するようになりました。単純に多くの人にプレイしてもらいたいなら、こちらをメインにするのがよいといっていいくらいです。

ただし iOS アプリに関しては、唯一のマーケットとなる App Store で 18 禁作品が禁止となっています。Android アプリに関しては、公式マーケットである Google Play では 18 禁作品は公開できませんが、中には業者が独自に運営している 18 禁専門のマーケットがありますので、興味のある方は探してみましょう。

●iOS アプリを制作する際の注意

様々な環境で制作とリリースができるのがメリットとはいえ、場合によってはそれなりに手間がかかります。特にリリースまでのハードルが高くなるのが、iOS アプリを作る場合です。

iOS アプリの制作には、Mac のパソコンが必要になります。そして iOS Developer Program という開発者向けプログラムに登録しなければなりません。この登録は有料で、年間 99 ドルかかります。為替レートによって日本円の価格は変動することがありますが、おおむね 1 万円になります。さらに App Store に作品を登録するまでのプロセスも、多少ややこしいものがあります。

これまで Windows しか使っていなかったけれど、iOS アプリにもチャレンジしたい……そんな人が多くなっていますが、制作環境の構築やコスト面でなかなか苦労することになりますので、そこを理解した上で取り組むようにしてください。

『LemoNovel AS3 の機能』

●作りやすく、多機能

LemoNovel AS3 はクリエイターにとっては作りやすく、ユーザーにとっては多機能な作りとなっています。基本的なテキスト表示や装飾、グラフィックの表示、フェード演出、サウンド再生、ムービー再生、変数管理、メッセージボックス表示、複雑な処理を簡素化するマクロ機能など、アマチュアゲームとしてはもちろん商業作品のエンジンとしても実用的です。

一般的なテキスト主体のノベル・アドベンチャーゲームだけでなく、コマンド選択を駆使した恋愛､戦闘シミュレーションタイプのゲームも可能でしょう。工夫次第で、様々なゲームを作ることができるはずです。

●自動アップデート機能

AIR 版限定の機能になりますが、作品をバージョンアップまたはバグ修正した際に、ユーザーが作者サイトにアクセスすることなく、最新版をダウンロードさせることができます。

これは Adobe AIR のシステムが本来持っている機能で､通常なら最新版をアップロードした上で自分のサイトに告知し、ユーザーにダウンロードしてもらわなければなりませんが、そういったプロセスが不要となるのです。ユーザーと制作者、双方にとって有益と思われます。

●多言語対応機能

ユーザーの言語設定に応じて、作品内のテキストを切り替えられる機能があります。

たとえば英語の翻訳版を出したいという時には、従来は日本語版と別にパッケージングしてリリースする必要があり、バグ修正などの対応も手間が倍になりましたが、LemoNovel AS3 なら必要な設定をするだけで、テキストのみを切り替えられます。

日本のユーザーのみを対象にしている場合は必要ありませんが、ワールドワイドでリリースできるiOSやAndroidアプリで、特に有効活用できるでしょう。世界中に日本のゲームファンはいるので、彼らの注目を集めたいなら必須の機能です。

●プラグイン機能

プラグインとは、そのシステムが元来持っていない機能を追加するプログラムのことで、LemoNovel AS3もこれに対応しています。

プラグインを作成するには、ActionScript3.0というFlash開発用のプログラミング言語（LemoNovel AS3の由来でもある）と専用のソフトを使わなければなりません。したがって基本的に上級者向けの機能になりますが、最初からこれを習得しているなら、あるいはスキルを持っていて協力してくれる人がいるなら、おおいにクオリティアップに役立つはずです。

●タッチ操作に対応

AS3版、AIR版、AIR Mobile版のすべてにおいてタッチ操作が可能です。スマートフォンはもちろん、今後増えると予想されるタッチスクリーンPCでの活躍が期待できます。

●制作はタグ記述式

一般にノベル・アドベンチャーゲームのシナリオは、テキスト本文と各種の命令・制御をするスクリプトで構成されます。スクリプトを書くことを「タグを記述する」とも言います。

```
[LoadBG surface='PRIMARY' path='./resource/BG/sampleBG1.jpg']
```

たとえば背景画像を表示する際はこのような書き方をします。ウェブサイトの制作に用いるHTMLと同じ感覚で、必要なのは最低限の算数や英語の知識だけです。他のタグ記述式の制作ツールやHTMLの経験者はもちろんですが、こういった言語に馴染みのない人でも比較的覚えやすいはずです。

『サンプルをダウンロード』

●実際に機能を確認する

LemoNovel AS3 のメリットやおおまかな機能がわかったところで、公式サイトにアクセスしましょう。

【LemoNovel AS3 公式サイト】
http://www.le-mo.jp/lemo/products/LemoNovel_AS3/index.htm

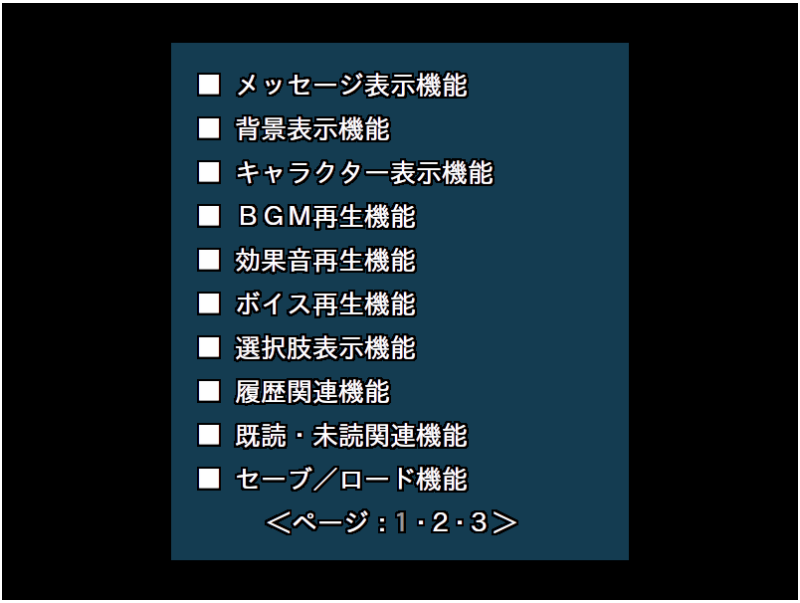
まずは［はじめに］［システム利用規約］［システム動作環境］をチェックして、それから［サンプル］のページを見てください。LemoNovel AS3 の機能を実際に試すことができます。

サンプルは AS3 版と AIR 版がありますが、ブラウザを起動する必要がなく、かつフォルダやファイルの構成をそのまま見られる AIR 版をおすすめします。なお、Adobe AIR がパソコンにインストールされていない場合は、サンプルのリンクをクリックすると Adobe AIR のインストーラーが起動するので、わざわざ Adobe のサイトにアクセスしてダウンロードする必要はありません。

インストール場所は「C:¥ Program Files (x86)¥LEMO¥Sample」で、Mac は「アプリケーション¥LEMO¥Sample」になります。フォルダのショートカット（Mac はエイリアス）を作成しておくとよいでしょう。

では、いよいよ生成されたフォルダの中にある実行ファイルを起動してください。Windows は exe、Mac では app と拡張子が違います。

画面が起動してしばらく読み進めると、［メッセージ表示機能］［背景表示機能］［キャラクター表示機能］といったメニューが出てきます。これらに目を通していってください。操作方法については公式サイトの［プレイ操作方法］で確認できます。



これらのサンプルがどのようなスクリプトで動き、どのような素材を使っているかは、script フォルダと resource フォルダ内のファイルでわかりますので、本書と平行して見るようにしてください。これらのフォルダの置かれている場所は、Windows と Mac で異なりますが、これについては第 2 章で解説します。

ちなみに AIR Mobile 版についてはサンプルがありませんが、公式サイトの［リンク］に AIR Mobile 版を採用した作品が紹介されています。サンプルとしてではなく完成されたゲームとして遊べますので、感触を確かめてみてください。

第 2 章

制作準備をしよう

『ファイル・フォルダ構成の把握』

●LemoNovel AS3 の構成要素

LemoNovel AS3 でゲームを作るにあたり必要なフォルダ･ファイル構成は、以下のようになっています。

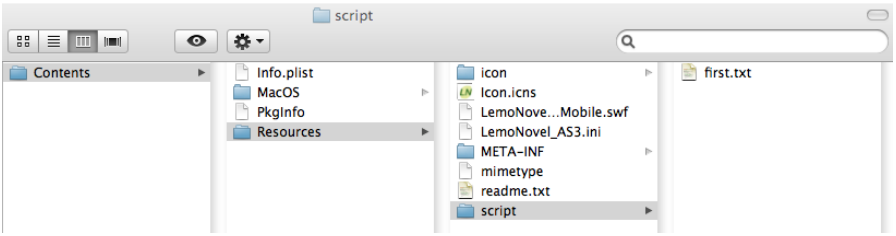
システム本体	
LemoNovel_AS3.swf LemoNovel_AIR.swf LemoNovel_AIR_Mobile.swf	LemoNovel AS3 の本体です。AS3 版は HTML ファイル、AIR 版と AIR Mobile 版は実行ファイルを起動することで動作します。
初期設定ファイル	
LemoNovel_AS3.ini	システムの初期設定を記述するテキストファイルです。
シナリオフォルダ	
名称は自由 ※サンプルでは script	シナリオとその他のテキストファイルを入れておくフォルダです。
素材フォルダ	
名称は自由 ※サンプルでは resource	グラフィックやサウンドなどの素材ファイルを入れておくフォルダです。
アイコンフォルダ	
名称は自由 ※サンプルでは icon	作品をリリースする時に使用するアイコンを入れておくフォルダです。AS3 版には含まれません。AIR 版と AIR Mobile 版でサイズが異なります。

AIR 版と AIR Mobile 版の場合、上記以外のファイルやフォルダがありますが、インストールの際に自動生成されるもので、制作者が編集することはありません。

●Mac で制作する場合

AIR 版と AIR Mobile 版において、Windows では各ファイル・フォルダが同じ階層に配置されるのに対し、Mac では app 形式としてひとつのファイルにまとめられる形となります。

これは Adobe AIR アプリケーションの仕様なのですが、中身を見るには右クリックから［パッケージの内容を表示］を選択します。ここで表示される［Contents］というフォルダの中にさらに［Resources］というフォルダがあり、その内部にシステム本体をはじめとするファイルとフォルダがあります。



したがって Mac で制作する場合は［Resources］フォルダのエイリアスを作っておくと便利です。

『扱えるファイルの種類』

●テキストファイルの扱い

LemoNovel AS3 で使用するテキストファイルのフォーマットは一般的な txt ですが、ひとつ注意点があります。それは、文字コードを必ず UTF-8（Unicode）形式にして保存しなければならないことです。

テキストファイルを新規作成した時に別の形式になっていたら、まず文字コードの変換をしてください。これを怠ると、ゲームが正常に表示されなくなります。初期設定ファイルの LemoNovel_AS3.ini や、後に説明する定義ファイルと呼ばれるテキストファイルも、同様に UTF-8 で保存する必要があります。

●素材ファイルの扱い

LemoNovel AS3 で使用する素材ファイルのフォーマットは、以下のとおりです。

素材の種類	フォーマット
グラフィック	JPG / PNG / SWF / GIF
サウンド	MP3
ムービー	Flash Player で再生可能な動画 / SWF
プラグイン	SWF

キャラクター用のグラフィックは、透過形式でなければいけません。PNG、SWF、GIF が該当しますが、もっとも一般的なのは PNG になります。ムービーの「Flash Player で再生可能な動画」とは、MP4 や FLV などのフォーマットのことです。

テストとして LemoNovel AS3 を触っていくという段階では、インターネット上のフリー素材を集めておきましょう。背景画像、キャラクター画像、BGM、効果音など、ムービー以外のたいていの素材が集まるはずです。

『制作用の本体をダウンロード』

●最初に素材フォルダを作成

サンプルの確認で操作に慣れたら、制作用の本体を手に入れましょう。公式サイトの［ダウンロード］から自分がリリースを予定している種類と画面サイズを選択して、ダウンロードしてください。AIR 版と AIR Mobile 版にあるパッケージ作成用については、今は無視して大丈夫です。

ダウンロードした本体には素材フォルダがありませんので、新規に作成してください。フォルダ名は自由ですが、サンプルにならってつけるのがわかりやすいです。

ルートフォルダ	サブフォルダ
resource	BG（背景画像） BGM（音楽） Button（ボタン画像） Char（キャラクター画像） Effect（エフェクト画像） Movie（動画） Sound（効果音） System（システム関係） Voice（キャラクターボイス）

必要ないと思ったサブフォルダについては、作成しなくても構いません。肝心の素材は、前述のようにフリー素材もいいですが、サンプルから流用してもよいでしょう。

さて、実行ファイルを起動してみると、簡単な案内メッセージだけが表示されます。script フォルダの中には first.txt がありますが、当面はこれを編集することで、タグの使い方を覚えていってください。

●セキュリティの設定を行う

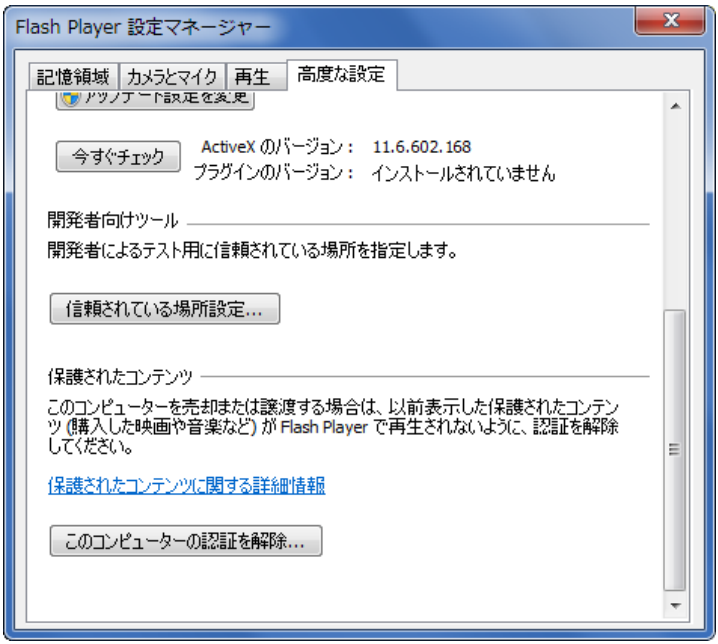
AS3 版でのみ発生する問題なのですが、Flash Player の仕様により、初期状態では HTML ファイルを実行しても何も起動しません。グラフィックやサウンドなど、各種ファイルにアクセスしようとするローカル環境の SWF ファイルが、セキュリティ上の問題を含んでいると見なされてしまうのです。

これを回避するためには、下記の手順で設定を変更します。

① Windows の場合、コントロールパネルの［Flash Player］を選択。Mac の場合システム環境設定の［その他］→［Flash Player］を選択。

② ［高度な設定］タブから［信頼されている場所設定］を選択。

③ ［追加］を選んで AS3 版のフォルダを指定する。



『シナリオファイルの書き方』

●コマンド名とパラメータ式

スクリプト中の各種命令を LemoNovel AS3 では「コマンド」と呼びます。コマンドは「コマンド名」と「パラメータ式」で構成されます。第 1 章で例に挙げたスクリプトをもう一度見てみましょう。

```
[LoadBG surface='PRIMARY' path='./resource/BG/sampleBG1.jpg']
```

このうち［LoadBG］がコマンド名で、『surface～』と『path～』がパラメータ式になります。パラメータ式は『【パラメータ名】=【値】』で構成されます。コマンド名とパラメータ式は、必ず半角の空白を挟んで記述してください。全角だとタグそのものが無視されてしまいます。

パラメータ式の値は、シングルクォーテーション（'）で囲む場合とダブルクォーテーション（"）で囲む場合がありますが、一部の記述を除いて、シングルクォーテーションで囲むようにしてください。囲まなくても動作はするのですが、処理の都合上、囲むことが推奨されます。

中には 50 以上のパラメータを持つタグもありますが、そのすべてを記述する必要はなく、省略できるパラメータが大半です。省略したパラメータは「無効」「現状維持」「デフォルト値」のいずれかで処理されます。

```
改ページを待ちます。[p]
```

このようにパラメータがないコマンドもあります。

コマンド名もパラメータ名も、大文字と小文字の区別は厳密に決められていますので、間違いがないように記述しなければなりません。もし思うように動作しない場合は、このあたりを疑ってください。

本書ではすべてのコマンドは紹介しきれないので、必要に応じて公式サイトの［タグリファレンス］のページをご覧ください。

●コメント機能

シナリオファイルに記述したスクリプト以外のテキストは、通常ではすべて画面に表示されてしまいます。そこでファイル内に注釈やメモを記述したい場合に、コメントという機能を使います。

```
// 初期設定
```

サンプルの first.txt の 1 行目はこのように記述されていますが、画面には表示されません。半角スラッシュを 2 個繋げた「//」を記述すると、記述位置以降のすべてのテキストが表示されなくなるのです。「//」という文字列自体を表示したい場合には、メッセージ出力専用のコマンドを使用することになります。

複数行をまとめてコメント化することも可能です。

```
/*
複数行を[r]
コメント化することもできます。[p]
*/
```

このように「/*」と「*/」の間に記述します。

『初期設定（ini）ファイル』

●ゲームの基本部分を設定する

シナリオファイルにスクリプトを記述するのと同時に、LemoNovel_AS3.iniも制作するゲームのスタイルに応じて編集していくことになります。

種別	概要
システム関連	もっとも基本となる設定
スクリプト関連	first.txt などスクリプトファイルの設定
メッセージレイヤー関連	メッセージ表示の位置やフォントの大きさなど、通常画面のメッセージの設定
履歴レイヤー関連	履歴画面のメッセージの設定
メッセージボックス関連	「はい」「いいえ」などを表示するメッセージボックスの設定
トースト関連	通知メッセージの設定
ヒント表示関連	リンクにマウスを合わせた時に表示されるヒントの設定
キーボード・マウス操作関連	ユーザーの操作に関する設定
カスタムポインタ関連	マウスポインタの調整に関する設定
オートモード関連	自動読み進みに関する設定
履歴関連	履歴画面ページ数などの設定
プラグイン関連	進捗表示画面などプラグインの設定

あまりいじらないだろう項目から、ゲームの根幹に関わる項目まで多数あります。特にメッセージレイヤー関連は頻繁に変更することになるでしょう。

実はこのファイルには、記述が省略されている項目があります。残りの項目については公式サイトの［初期設定（ini）ファイル］で確認し、必要に応じて自由に追加していってください。

●ゲーム ID を編集する

まず編集しなければならないのは、システム関連の[game_Id]という項目です。オリジナルの ID であることが推奨されますので、他の作品と被らない、かつわかりやすい文字列にしましょう。

候補としては次のようなものが考えられます。

【ゲームタイトルをローマ字に変換】

たとえば『最強物語』なら［Saikyou_Monogatari］。

長いタイトルならば適当に略す。

【制作者名＋連番】

アライという制作者名なら［Arai_0001］。

作品が増えるたびに 0002、0003、0004……としていく。

もっとも、最終的なリリースまでに決定すればいいので、初めて触るという段階では「test」とでもしておくのがよいでしょう。

『コラム 1・スマホアプリのダウンロード数』

私自身がそうだったのですが、LemoNovel AS3 を使おうと思った方は、Android、iOS 向けアプリを作ってリリースしたいという目標を持っているのではないかと思います。普通に Windows 向けを作るなら、他にも選択肢があるわけですから。

そこで気になるのが、一個人が（あるいは一サークルが）スマホアプリをリリースしたところで、どれほどダウンロードされるのかということです。無料であれば､数十万ものダウンロード数を叩き出している同人ゲームもあります。しかしこれは特別な例と考えたほうがよさそうです。Android アプリのマーケットである Google Play、iOS アプリのマーケットである App Store、いずれも新作が毎日のように大量追加され、ほとんどのアプリは埋もれていく運命にあります。

Android、iOS の両方でリリースしたことがある私の場合ですが、ほとんど宣伝をしていないという理由もあるのですが、1 日にせいぜい数回のダウンロード数です。有料アプリも出していますがこちらは 1 ヶ月に数回程度です。本気で大勢に遊んでもらいたいと思うなら、しっかり宣伝はしましょう。

第 3 章
スクリプトを構築しよう・シナリオ編

『テキストを表示する』

●テキスト表示の基本

初めて first.txt を開くと、次の記述があります。これはそのままテキスト表示の基本となっているのです。

```
//ここからスクリプトを書き始めてください。

[MsgLayer visible=true]

この本体にはサンプルは含まれていません。[l][r]
サンプルをご覧になりたい方は、サンプル付きの方をダウンロードしなおしてください。
```

まず[MsgLayer]はメッセージレイヤー全般の設定をするコマンドです。最初にこのコマンドのパラメータ『visible』を『true』に指定することで、メッセージレイヤーが表示されます。逆に『false』を指定すると非表示状態になります。非表示状態では、メッセージレイヤーだけでなくその上のテキストも消えることになります。なお、『true』にしたあとで再び[MsgLayer]コマンドを使用する場合、『visible』は省略しても『true』と処理されます。以後の記述例でも[MsgLayer]コマンドの『visible』は省略しますが、『true』であるという前提で見てください。

次に[l]はクリック待ち（AIR Mobile 版はタップ待ち）をするコマンドです。進行が停止している間、メッセージレイヤーにクリック待機アイコンが表示されます。そして[r]は改行をするコマンドです。LemoNovel AS3 では、ファイル上でテキストの改行をしても画面上では反映されませんので、改行したい時は基本的に[r]コマンドを用いることになります。

```
次に２行空けます。[l][r]
[r]
[r]
２行空けました。[p]
```

このように[r]を連続することで、任意の行数を空けることもできます。HTML でウェブサイトを作ったことのある人にとっては、馴染み深い手法だと思います。

最後に出てきた[p]は、改ページをするためのコマンドです。これを記述すると、メッセージレイヤー上のすべてのテキストが消えます。視覚的に、次のページに進んだという処理になるのです。

●座標とサイズを調整する

画面サイズによって細かく数値は違いますが、メッセージレイヤーの座標とサイズは、アドベンチャー形式に適した形になっています。つまり画面下部に数行だけが収まる形です。

この座標とサイズの初期位置は初期設定（ini）ファイルで決まっていますが、シナリオの途中で変えたい場合には次のようにします。

```
[MsgLayer posX='30' posY='30' width='740' height='540']
```

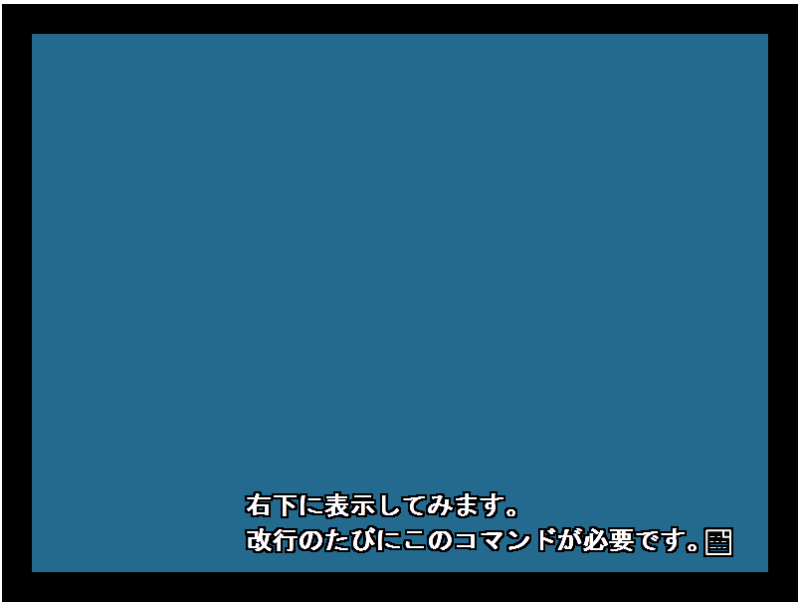
左上（0,0）を基点として、『posX』は X 座標、『posY』は Y 座標の値です。『width』は横幅、『height』は縦幅になります。

座標もサイズもそう頻繁に変更することはないと思いますが、通常シーンではアドベンチャー形式で、回想シーンでは全画面のノベル形式にするなど、使い方によってはよい演出効果が期待できます。

●テキストの位置を変更する

メッセージレイヤー内のテキストは、通常は左上から流れるようになっていますが、自由に表示位置を決めることができます。

```
[Locate posX='270' posY='440']
右下に表示してみます。[r]
[Locate posX='270' posY='475']
改行のたびにこのコマンドが必要です。[p]
```



この[Locate]コマンドを使えば、あらゆる位置にテキストを配置できます。様々な応用が利くテクニックです。

気をつけなければならないのは、[Locate]コマンドによる設定が反映されるのは直後の 1 行だけで、改行または改ページすると、元に戻ってしまいます。

もうひとつ、メッセージレイヤー全体でテキストの位置を調整できます。

```
[MsgLayer align='CENTER']

中央に表示されます。[r]
変更しない限りずっとそのままです。[p]
```

[MsgLayer]コマンドのパラメータ『align』には、『LEFT』『CENTER』『RIGHT』の 3 種類の値があります。デフォルトでは『LEFT』ですが、『CENTER』で中央揃え、『RIGHT』で右揃えにできます。[Locate]コマンドと違うのは、もう一度『align』を変更しない限り、その設定は保持されることです。

ところで実際にこのスクリプトを試すとわかるのですが、中央揃えや右揃えだと、テキストが画面に出現する際に「中央から左右に広がる」「右から左に広がる」という、人間の目には見慣れない現象が起きます。これは瞬間表示することで解決できます。

```
[MsgLayer align='CENTER' speed_Normal='0']

中央に瞬間表示されます。[l]

[MsgLayer align='LEFT' speed_Normal='USER']

左揃えに戻し、表示速度もデフォルトに戻りました。[p]
```

『speed_Normal』はメッセージ表示速度を指定するパラメータです。0 にすると瞬間表示になり、『USER』を指定すると、元の速さに戻ります。

ちなみに「中央に瞬間表示されます。」の末尾に[r]コマンドを使用していませんが、切り替え時点の行に 1 文字以上ある場合に『align』で位置揃えを変更すると、自動的に改行が行われるようになるのです。

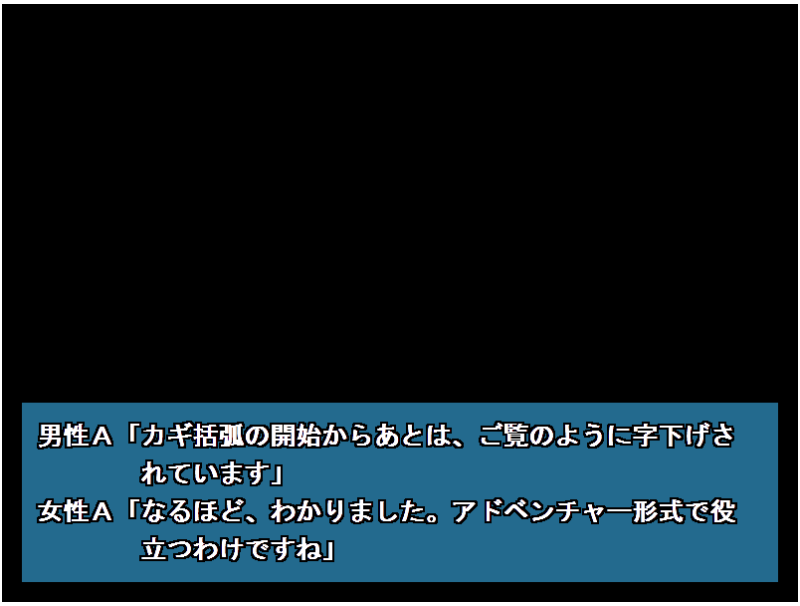
●インデント機能を使う

インデントとは字下げのことで、ワープロソフトなどには基本機能として搭載されていますが、LemoNovel AS3 では簡単な記述でこれを実現できます。アドベンチャー形式ではこれが役立ちます。

```
男性Ａ「[Indent]カギ括弧の開始からあとは、ご覧のように字下げされています」[EndIndent][r]
女性Ａ「[Indent]なるほど、わかりました。アドベンチャー形式で役立つわけですね」[EndIndent]
```

[Indent]コマンドでインデントが開始され、[EndIndent]コマンドで終了します（代わりに[p]と改ページしても終了します）。

画面で見るとこのようになります。



『テキストを装飾する』

●フォントを変更する

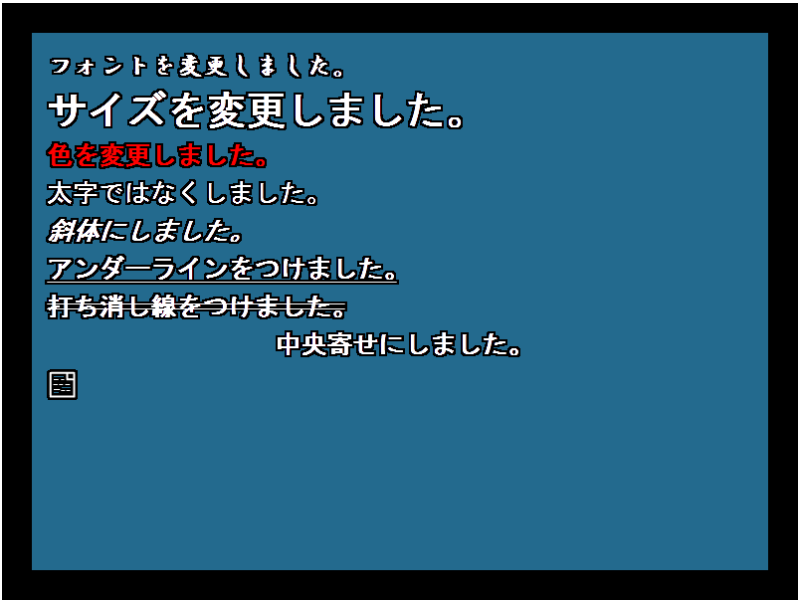
状況によって、テキストを様々に工夫して表示したいことがあります。そういった時には、[Font]コマンドでたいていのことが可能です。

このコマンドで使えるパラメータは種類が多いのですが、代表的なものを以下に列挙します。

パラメータ名	概要
name（フォント名）	フォント名を指定
size（数値）	サイズを指定
color（RGB 値）	色を指定
bold（true / false）	ボールド体（太字）表示を指定
italic（true / false）	イタリック体（斜体）表示を指定
underline（true / false）	アンダーライン表示を指定
strike（true / false）	打消し線表示を指定
align（LEFT / CENTER / RIGHT）	メッセージの位置揃えを指定

では、実際にこれらを試してみましょう。終了コマンドの[/Font]とセットで使うことに注目してください。

```
[Font name='HG 行書体']フォントを変更しました。[/Font][r]
[Font size='40']サイズを変更しました。[/Font] [r]
[Font color='0xff0000']色を変更しました。[/Font] [r]
[Font bold='false']太字ではなくしました。[/Font] [r]
[Font italic='true']斜体にしました。[/Font] [r]
[Font underline='true']アンダーラインをつけました。[/Font] [r]
[Font strike='true']打ち消し線をつけました。[/Font] [r]
[Font align='CENTER']中央寄せにしました。[/Font][p]
```



フォントを変更する『name』は、ユーザーのパソコンに指定したフォントがインストールされていなければ機能しません。確実に意図したフォントを表示させたい場合は埋め込みフォントを使用しますが、これについては第 4 章で解説します。

サイズを変更する『size』は、単純に数値を指定します。キャラクターが大声を上げたなど、ここぞという時に使えばユーザーを驚かせられるでしょう。

色を変更する『color』は、パソコンで色を表現する際に使う RGB 値という値で指定します。どの値がどの色に対応しているかは、検索すればすぐに出てきますのでチェックしてみてください。

『bold』『italic』『underline』『strike』の 4 つは、それぞれ『true』か『false』で指定します。いずれもちょっとしたアクセントとして効果を発揮することでしょう。

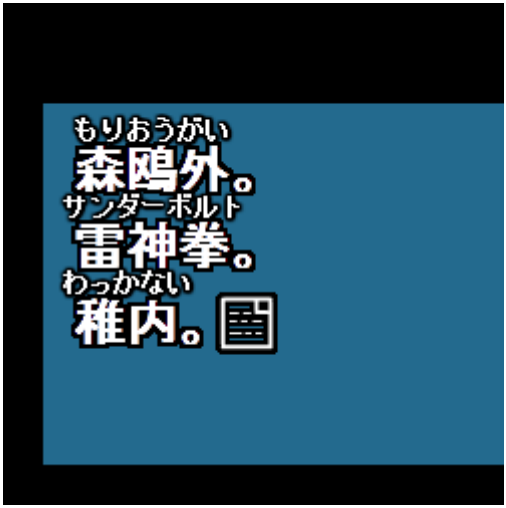
そして『align』は、[MsgLayer]コマンドと同様の機能です。この行だけ中央寄せにしたい、という時に使います。

●ルビを振る

シナリオを書いていると、難しい読みの漢字が出てくることがあります。ユーザーに確実に読ませたいなら、ルビを振っておきましょう。難しい読みでなくても、キャラクター名や地名などの固有名詞については、初出の時にルビを振るのが得策です。また、ライトノベル風の作品では、必殺技に特徴的なルビを振ることもよくあります。

```
森[Ruby text='もり']鴎[Ruby text='おう']外[Ruby text='がい']。[r]
雷神[Ruby text='サンダーボルト']拳。[r]
稚内[Ruby text='わっかない' align='BEF_SPC']。[p]
```

ルビを振るには[Ruby]コマンドを使用します。以下が実際の画面です。

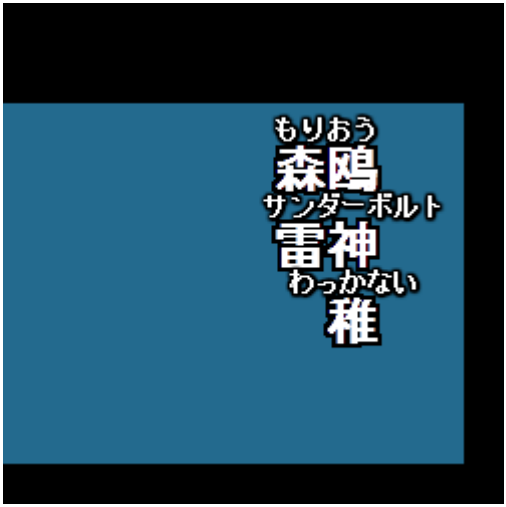


[Ruby]コマンドを記述すると、その直前の文字に対し、中央寄せでルビが振られます。「森鴎外」の例では、一字ずつルビを振る基本的な使い方をしています。

ちょっと工夫が必要なのが、「雷神拳」のような長いカタカナのルビを振りたい場合です。ルビは中央に揃うのが一番見栄えがいいので、この例では真ん中の 2 文字目にすべてのルビを集中させています。すると見事に単語全体にルビがかかっているように見えるのです。

しかし単語の文字数が偶数の場合、この方法は使えません。そこで『align』のパラメータを『BEF_SPC』に指定します。これは「直前の文字の、そのまた直前のスペース」に中央揃えで合わせるという機能です。上記の例では「稚」と「内」の間のスペースということになります。

一文字ずつルビを振る、あるいは一ヶ所に集中させる。状況に応じて使い分けるのがいいのですが、注意点として、ルビは折り返し機能がありません。



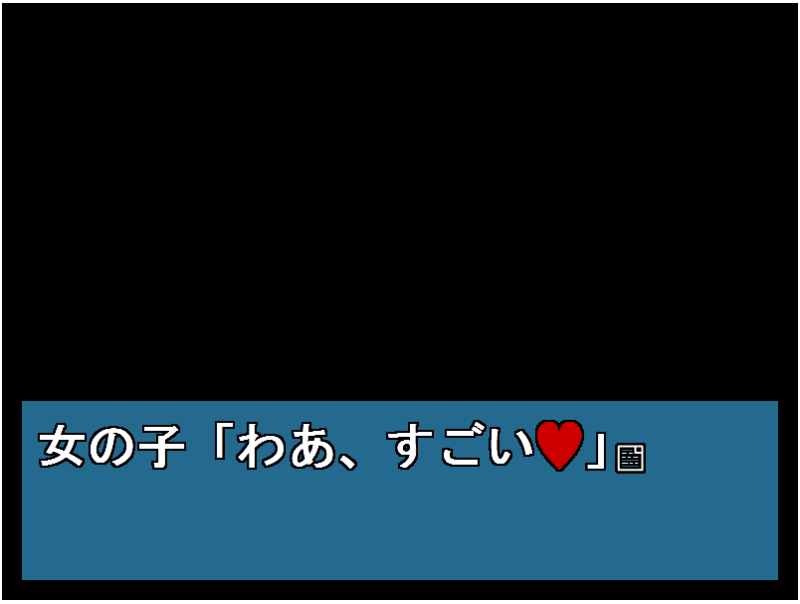
テキストはその行の終端に達すると自動的に折り返されますが、ルビを一ヶ所に集中させる方法では、正常に表示されないのです。ノベル・アドベンチャーゲームに限らず、小説でもこうしたルビの振り方はされていないはずです。これを防ぐには、前後の文字を調節するしかありません。こうしたミスをしたままリリースしないように、注意深くチェックしましょう。

ちなみに[Ruby]コマンドには、[Font]コマンドと同様に様々な装飾用パラメータがあります。必要と思ったら使ってみてください。

●絵文字を使用する

テキストファイルには入力できない、たとえばハートマークなどの記号を表示したい場合があります。これは絵文字表示の[PicChar]コマンドで実現できます。

```
女の子「わあ、すごい[PicChar  path='./resource/Char/heart.png']」[p]
```



今後の画像関連コマンドでも使っていくのですが、『path』パラメータは画像ファイルのパス（置き場所）を指定します。この例では、

「実行ファイルと同じ階層にある resource フォルダ、その中にある Char フォルダ、その中にある heart.png ファイル」

を意味します。

絵文字をふんだんに使う携帯メールを再現してみるとか、普通の文字では表現できない怪物の叫び声を演出してみるとか、メッセージをさらに彩ることが可能です。なお、この機能で出力された画像はテキスト扱いになるので、履歴画面にも表示されます。

『背景画像を表示する』

●背景画像表示の基本

制作している本体のサイズに合わせた背景画像は用意できているでしょうか。BG フォルダに入れたら、以下のスクリプトを試してみましょう。

```
[BGLayer visible='true']
[LoadBG surface='PRIMARY' path='./resource/BG/bg01.jpg']

背景画像を表示しました。[p]
```

背景画像を表示するためには、最初に[BGLayer]コマンドを記述し、『visible』を『true』にする必要があります。

そして背景画像を読み込むのが[LoadBG]です。『path』は前項の絵文字で解説したとおりですが、『surface』は画像を読み込むサーフェスを指定するパラメータです。サーフェスとは日本語で「面」などと訳されます。背景画像には 2 つのサーフェスがあり、それぞれプライマリサーフェスとバックサーフェスといいます。

プライマリサーフェスは、いわば表の面です。したがって『PRIMARY』と指定すれば背景画像が画面に現われます。表示は瞬間的です。これが背景画像表示の基本となります。

しかし実際のゲームをプレイすればわかりますが、背景を瞬間表示で切り替えることはあまりありません。加えて LemoNovel AS3 の仕様なのですが、『PRIMARY』で表示すると切り替えに伴い一瞬だけ黒い画面が見えてしまいます。そこで「何らかの演出効果を付加しながらスムーズに切り替えていく」という方法を採用していくことになります。

●エフェクトをつけて背景画像を表示する

もうひとつのサーフェス、バックサーフェスはすなわち裏の面です。ここに読み込んでも、背景画像は何も表示されません。試しに先程の例の『PRIMARY』を『BACK』に書き換えてみてください。

ではバックサーフェスの用途は何なのかというと、プライマリサーフェスに切り替えるための準備場所です。画像をいったんバックサーフェスに読み込んでおいて、それからエフェクトの種類と時間を指定して、プライマリサーフェスと入れ替える……これが LemoNovel AS3 における背景画像表示のスタンダードなのです。

```
[LoadBG path='./resource/BG/bg01.jpg']
[SetFixedBGTrans mode='CROSSFADE' time='1000']
[StartTrans]

背景画像をクロスフェードで表示しました。[p]
```

あれ？　と思った方がいるでしょうが、[LoadBG]コマンドの『surface』に『BACK』を指定していません。実は『surface』のパラメータは省略すると値が『BACK』と認識されるので、これでよいのです。

さて、背景画像のエフェクトを設定するのが[SetFixedBGTrans]コマンドです。『mode』にエフェクトの種類を指定し、『time』にどれほどの時間をかけて切り替えるかをミリ秒（1000 が 1 秒）で指定します。そして[StartTrans]コマンドでエフェクトを実行します。

エフェクト名	概要
CROSSFADE	混ざり合うようにして切り替わります。
SCROLL	画面外からスクロールして切り替わります。
FLIP	瞬間的に切り替わります。
MOSAIC	モザイク状に切り替わります。
NOISE	ノイズ交じりに切り替わります。
PIXEL_DISSOLVE	ピクセル単位で切り替わります。
UNIVERSAL	ルールファイルにしたがって切り替わります。
MASK_ANIM	SWF アニメーションにしたがって切り替わります。
FADEOUT_BLACK	徐々に黒い画面に切り替わります。
FADEOUT_WHITE	徐々に白い画面に切り替わります。

一番汎用性が高いのが『CROSSFADE』でしょう。これだけ使っても、最低限ノベル・アドベンチャーゲームとして形にはなります。

それでは『CROSSFADE』以外についても例を挙げていきます。

■SCROLL

```
[SetFixedBGTrans mode='SCROLL' dir='LEFT' stay='true' time='1000']
```

紙芝居のように現われる『SCROLL』は、別途『dir』というパラメータが必要です。どの方向にバックサーフェスの画像がスクロールされるかを指定するのです。『dir』が『UP』なら上方向、『DOWN』なら下方向、『LEFT』なら左方向、『RIGHT』なら右方向にスクロールします。

また『stay』というパラメータは、『true』ならプライマリサーフェスがその場に留まります。つまりバックサーフェスがかぶさるようにスクロールしてきます。『false』ならバックサーフェスに押し出されるようにして画面外に消えていきます。省略すると『false』になります。

■FLIP

```
[SetFixedBGTrans mode='FLIP']
```

瞬間表示したい時に使うのが『FLIP』です。急にライトを向けられた、などのシーンで使えるでしょう。このエフェクトは、必然的に『time』パラメータを指定する必要がありません。

■MOSAIC

```
[SetFixedBGTrans mode='MOSAIC' time='1000' mosaicSize='80']
```

モザイクで切り替える『MOSAIC』には、1 マスの幅を指定する『mosaicSize』パラメータがあります。この値の下限は 8 で、上限は画面の横幅になります。省略すると、その横幅の 4 分の 1 が自動的に入ります。

■NOISE

```
[SetFixedBGTrans mode='NOISE' time='1000' colorCh='RED|BLUE']
```

サスペンス風にノイズで切り替えるのが『NOISE』です。『colorCh』パラメータは、ノイズの生成に用いるカラーを指定します。『RED』『GREEN』『BLUE』から選ぶのですが、2 つ以上を連結することもできます。『RED|BLUE』ならば赤＋青で紫色のノイズになるのです。省略すると、色のない普通のノイズになります。

この例では省略していますが、灰色で表示する『grayScale』というパラメータもあります。

■PIXEL_DISSOLVE

```
[SetFixedBGTrans mode='PIXEL_DISSOLVE' time='1000']
```

『CROSSFADE』が画面全体がいっぺんに切り替わっていくのに対して、『PIXEL_DISSOLVE』はピクセル（画面を構成する最小単位）のひとつひとつがバラバラに切り替わっていきます。他に指定するのは『time』だけです。

■UNIVERSAL

```
[SetFixedBGTrans mode='UNIVERSAL' path='./resource/Effect/wipe_lr.png' vague='127' time='1000']
```

ユニバーサルエフェクトと呼ばれる独自のエフェクトを実現するのが『UNIVERSAL』です。これを使用するには、ルールファイルという白黒のグラデーション画像を媒体にします。サンプルの Effect フォルダに入っているので流用しましょう。

これを使用すると、黒から白、つまり左から右にバックサーフェスが現われてきます。ルールファイルは制作者がそれぞれ自由に作れるので、画面切り替え方法の幅がグッと広がります。『vague』は切り替えの境界のボカし具合を決めるパラメータです。0 から 255 で指定し、値が小さいほど境界はくっきりして、大きいほどボカされます。省略した場合は 64 になります。

ルールファイルはちょっと多機能なグラフィック編集ソフトであれば、グラデーション機能がついていますので、それで作成しましょう。なお、形式は JPG よりも画像劣化のない PNG で作成することが推奨されます。

■MASK_ANIM

```
[SetFixedBGTrans mode='MASK_ANIM' path='./resource/Effect/mask_ball.swf' maskSurface='BACK' time='2000']
```

『MASK_ANIM』は透明な部分と不透明な部分で構成した SWF アニメーションを使うことで、『UNIVERSAL』よりも独特のエフェクトを見せられます。こちらもサンプルから流用して試してみましょう。『maskSurface』のパラメータの値は『PRIMARY』『BACK』があり、それぞれで効果が違ってきます。

使いこなせれば楽しいですが、SWF アニメーションを作るところから始めなければならず、そのスキルがない人には縁のないエフェクトになります。

■FADEOUT_BLACK、FADEOUT_WHITE

```
[SetFixedBGTrans mode='FADEOUT_BLACK' time='4000']
[SetFixedBGTrans mode='FADEOUT_WHITE' time='4000']
```

黒でフェードアウトするのが『FADEOUT_BLACK』で、白でフェードアウトするのが『FADEOUT_WHITE』です。場面転換に用います。

このエフェクトはバックサーフェスに画像を読み込む必要がないのですが、『CROSSFADE』で黒い背景や白い背景を読み込んでも、ほぼ同じ効果が出ます。比べてみると微妙に違いますので、好きなほうを選ぶとよいでしょう。

『キャラクター画像を表示する』

●背景画像と違って ID が必要

キャラクター画像の表示は、背景画像の表示とそう変わりませんが、1 点だけ明確な違いがあります。

```
[LoadChar id='0' surface='PRIMARY' path='./resource/Char/01.png']

キャラクターを表示しました。[p]
```

キャラクターレイヤーを表示する[LoadChar]コマンドですが、[LoadBG]にはなかったパラメータ『id』に注目してください。1 枚だけ表示すればいい背景と違って、キャラクター画像は複数表示するのが前提です。

たとえば画面中央に表示するキャラクターは『0』にして、左なら『1』にするなど、『id』はキャラクターレイヤーID を管理するパラメータなのです。

ID は初期状態では 0、1、2 の 3 つを使えます。必要に応じて 999 まで増やすことができるのですが、それには[AddObj]というコマンドを記述します。

```
[AddObj type='CHAR' id='3']

キャラクターレイヤーＩＤに３を追加しました。[p]
```

追加するレイヤーまたはオブジェクトの区分を指定する『type』に『CHAR』を指定し、『id』に追加したい値（任意の文字列でも可）を指定します。

キャラクターレイヤーを増やすと、特にモバイル環境では動作が重くなることが予想されますので、不必要に増やすことは避けてください。

●キャラクターの表示と消去

キャラクター画像も背景画像と同じく、いったんバックサーフェスに読み込んでプライマリサーフェスに切り替えるという手法をメインにします。

```
[LoadChar id='0' path='./resource/Char/01.png']
[SetFixedCharTrans id='0' mode='FADEIN' time='500' posX='250' posY='
0']
[StartTrans]

キャラクター画像をフェードインで表示しました。[p]
```

キャラクターレイヤーのエフェクトを制御する[SetFixedCharTrans]コマンドを用います。まず『id』を必ず[LoadChar]コマンドで指定したのと一致させます。『mode』に指定した『FADEIN』は、キャラクター画像特有のエフェクトです。そして『time』『posX』『posY』をそれぞれ指定すれば、自分の好きな時間と位置でフェードイン表示させられます。

なお『mode』は、一部を除いて背景画像表示と同じ指定ができます。『FLIP』なら瞬間表示できますし、『MOSAIC』や『PIXEL_DISSOLVE』なら奇妙な感じに出現させることができるでしょう。

次は表示させたキャラクターを消去してみます。

```
[SetFixedCharTrans id='0' mode='FADEOUT' time='500']
[StartTrans]

キャラクター画像をフェードアウトで消去しました。[p]
```

『mode』を『FADEOUT』にしただけですが、これで指定した ID のキャラクターをフェードアウトできます。

ただしこの処理は、視覚的には目に見えない状態ですが、システム上で非表示になったわけではありません。どういうことかというと、表示か非表示かを指定する『visible』が『true』の値を保持しているのです。そのため、明確に非表示とする場合は[CharLayer]というコマンドを使用します。

```
[CharLayer id='0' surface='PRIMARY' visible='false']
```

この記述をすることで、キャラクターを瞬間的に消去することができます。応用すれば、『MOSAIC』などのエフェクトでも消去できます。

```
[CharLayer id='0' surface='BACK' visible='false']
[SetFixedCharTrans id='0' mode='MOSAIC' time='1000']
[StartTrans]
```

バックサーフェスを非表示と指定することで、その「非表示状態になったレイヤー」がエフェクトを伴いながらプライマリサーフェスに反映される、という流れになります。

●背景レイヤーとキャラクターレイヤーを同時に処理する

背景を切り替えてからキャラクターを表示する、あるいはキャラクターを消去してから背景を切り替える……こういった手順では、ゲームのテンポが悪くなる場合があります。そこでこの 2 つの処理を同時に行ってみましょう。

```
[MsgLayer visible='false']
[LoadBG path='./resource/BG/bg01.jpg']
[LoadChar id='0' path='./resource/Char/01.png']
[StartFixedStageTrans mode='CROSSFADE' time='2000']
[MsgLayer visible='true']

背景とキャラクターを同時に表示しました。[p]

[MsgLayer visible='false']
[LoadBG path='./resource/BG/bg02.jpg']
[CharLayer id='0' surface='BACK' visible='false']
[StartFixedStageTrans mode='CROSSFADE' time='2000']
[MsgLayer visible='true']

背景切り替えと同時にキャラクターを消去しました。[p]
```

背景レイヤー、キャラクターレイヤー、メッセージレイヤーを合成して、ひとつのステージとして処理するのが[StartFixedStageTrans]コマンドです。

1 回目では背景とキャラクターの両方をバックサーフェスに読み込み、それをひとまとめにしてクロスフェード表示させています。2 回目も新しい背景と非表示状態キャラクターを用意し、同様にクロスフェードさせています。このコマンドでは設定とエフェクト開始を同時に行うため、[StartTrans]コマンドは使用しません。また、切り替え時にメッセージレイヤーが表示されていると見栄えが悪いため、直前に[MsgLayer]コマンドを用いて非表示にしています。

『メッセージウィンドウを変更する』

●『path_BG』のパラメータ

初期設定では半透明の長方形をしたメッセージレイヤーですが、このままではどうにも味気ないものです。そこで画像を使ってみましょう。

初期設定（ini）ファイルを編集することで、メッセージレイヤーに背景画像を指定できますが、シナリオ内で指定するには[MsgLayer]コマンドを使用します。

```
[MsgLayer  path_BG='./resource/System/msgwindow.png']

メッセージレイヤーを画像にしました。[p]
```

System フォルダに画像を入れたら、『path_BG』のパラメータに、背景やキャラクターの要領で画像パスを指定します。これだけでメッセージレイヤーが画像になります。

しかしこの指定をするだけでは、場合によっては不十分です。メッセージレイヤーには透明度が設定されていますが、画像を使用していてもこれが反映されるのです。

初期設定（ini）ファイルの「メッセージ表示枠のアルファ値」という項目を見てください。アルファ値がつまり透明度で、デフォルトでは 0.7 となっています。小さいほど透明で、0 ならまったく見えなくなります。1 なら完全に不透明です。不透明にするか、ある程度の透明度を持たせるか、よく考えて決めましょう。なお、画像そのものに透明度をつけているのなら、この設定は必要ありません。

●メッセージウィンドウにキャラクターを表示する

アドベンチャーゲームではしばしば、メッセージウィンドウにキャラクターの顔グラフィックを表示して、誰がしゃべっているのかを明確にします。

通常、すべてのグラフィックはメッセージウィンドウの後ろに表示されるのですが……。

```
[CharLayer id='2' posX='20' posY='400' overMsg='true' syncHide='true']
[LoadChar id='2' surface='PRIMARY' path='./resource/Char/face1.png']
【女性】[r]
「左に私の顔が表示されているわ」[p]
```

[CharLayer]コマンドの『overMsg』パラメータを『true』にすることで、指定した ID のグラフィックを全メッセージレイヤーより前に表示させることができます。『syncHide』は、メッセージレイヤーを一時的に消去したときに、同時にグラフィックも消去するかの指定です。顔グラフィックはメッセージレイヤーの一部なので、通常は『true』にしましょう。

顔グラフィックはエフェクトが必要ないので、最初からプライマリサーフェスに瞬間表示するようにします。座標についても、[LoadChar]コマンドではなく[CharLayer]コマンドで指定します。

また、顔グラフィックとテキストが被らないように、初期設定（ini）ファイルで左側のマージン（余白）を調整してください。もし顔グラフィックを一時的に使わないシーンが必要になって、このマージンがいらないという場合は、[MsgLayer]コマンドで調整します。

```
[MsgLayer  margin_Left='15']
```

マージン調整は『margin_Left』が左、『margin_Right』が右、『margin_Top』が上、『margin_Bottom』が下となります。

『BGM を再生する』

●BGM 再生の基本

本項からはサウンド再生について取り上げていきますが、タグの書き方ではグラフィックとそう大差ありません。まずは次の例を見てください。

```
[LoadBGM buffer='PRIMARY' path='./resource/BGM/bgm01.mp3']
[PlayBGM buffer='PRIMARY' mode='PLAY']

ＢＧＭを再生しました。[p]
```

LemoNovel AS3 におけるサウンド再生は「読み込み→再生」というプロセスに分かれています。BGM の場合は[LoadBGM]コマンドで BGM フォルダのファイルを読み込み、[PlayBGM]コマンドのパラメータ『mode』で『PLAY』を指定します。双方のコマンドにある『buffer』は、グラフィックの『surface』に該当するパラメータです。ここで指定した『PRIMARY』は、再生のための表の領域ということになります。

さて BGM は普通､繰り返し再生することが前提ですが､歌曲などの場合は、1 回のみ再生したほうが好ましいこともあります。

```
[LoadBGM buffer='PRIMARY' path='./resource/BGM/song01.mp3']
[PlayBGM buffer='PRIMARY' mode='PLAY' repeat='0']

歌を１回だけ再生します。[p]
```

このように『repeat』パラメータに繰り返す回数を指定すればよいのです。再生する回数ではないので、繰り返さないなら 0 になります。

●エフェクトを用いて再生する

もう想像がついたと思いますが、『buffer』に『BACK』を指定（または省略）することで、グラフィックのようにエフェクトをつけながら再生することができます。

```
[LoadBGM path='./resource/BGM/bgm01.mp3']
[SetFixedBGMTrans mode='FADEIN' time='1000']
[StartTrans]

フェードしながらＢＧＭ再生します。[p]

[LoadBGM path='./resource/BGM/bgm02.mp3']
[SetFixedBGMTrans mode='CROSSFADE' time='1000']
[StartTrans]

クロスフェードでＢＧＭを切り替えました。[p]
```

サウンド再生におけるエフェクトは、フェードインとクロスフェードの 2 種類があります。[SetFixedBGMTrans]の『mode』パラメータに『FADEIN』か『CROSSFADE』を指定し、『time』パラメータに音量が最大になるまでの時間を指定します。そしてグラフィックと同じく[StartTrans]コマンドでエフェクトをスタートするのです。

また、このコマンドでも『repeat』パラメータで再生回数を決めることができますので、やはり歌曲などの場合は 0 に指定してください。

単純に再生するか、フェードインやクロスフェードで再生するか、どちらがいいかは状況によります。いずれにしても細かな雰囲気の違いを出すことができるでしょう。

●BGM を停止する

次は再生中の BGM を停止してみましょう。停止でも使用するコマンドは[PlayBGM]です。

```
[PlayBGM buffer='PRIMARY' mode='STOP']

ＢＧＭを停止しました。[p]
```

ただ停止するだけなら『mode』を『STOP』にすればいいのですが、ブツ切りに聞こえてしまうので実際には使いどころが難しいと思います。

したがって、BGM の停止はフェードアウトを基本とします。

```
[SetFixedBGMTrans mode='FADEOUT' time='1000']
[StartTrans]

ＢＧＭをフェードアウトで停止しました。[p]
```

再生時とほとんど変わりはなく、『mode』を『FADEOUT』と指定するだけです。フェードアウトの時間は、1～2 秒ほどが適切でしょう。

『効果音を再生する』

●効果音再生の基本

効果音を使用するには、BGM にはない ID が必要となります。背景画像とキャラクター画像の関係に似ているのです。

```
[LoadSound id='0' path='./resource/Sound/se01.mp3']
[PlaySound id='0' mode='PLAY']

効果音を再生しました。[p]
```

BGM のように[LoadSound]コマンドで読み込み、[PlaySound]コマンドで再生しますが、『id』パラメータに個別の ID を割り当てます。

```
[LoadSound id='0' path='./resource/Sound/se01.mp3']
[LoadSound id='1' path='./resource/Sound/se02.mp3']
[LoadSound id='2' path='./resource/Sound/se03.mp3']
[PlaySound id='0' mode='PLAY' repeat='1']
[PlaySound id='1' mode='PLAY' repeat='2']
[PlaySound id='2' mode='PLAY' repeat='INFINITY']

３種類の効果音を同時再生しました。[p]
```

これで複数の効果音を同時に再生することができます。部屋で家具がガタガタ鳴っている……などのシーンで使えるでしょう。効果音はデフォルトでは 1 回しか鳴らないので、繰り返し再生をするには『repeat』パラメータでその回数を指定してください。『INFINITY』はエンドレスで再生されます。

●効果音にもエフェクトをつけられる

エフェクトをつけて再生・停止できるのは効果音も同じです。

```
[LoadSound id='0' path='./resource/Sound/se01.mp3']
[SetFixedSoundTrans id='0' mode='FADEIN' time='1000' repeat='INFINITY']
[StartTrans]

フェードしながら効果音再生します。[p]

[SetFixedSoundTrans id='0' mode='FADEOUT' time='1000']
[StartTrans]

効果音をフェードアウトで停止しました。[p]
```

効果音のエフェクト[SetFixedSoundTrans]コマンドです。『mode』パラメータに『FADEIN』を指定すればフェードイン再生、『FADEOUT』を指定すればフェードアウト停止できます。

ただ、元々の効果音が数秒程度の再生時間では、エフェクトをかけたところで効果が薄いので、ある程度再生時間が長く、かつエンドレスで聞かせるのが前提の効果音で使うのがいいでしょう。以下に挙げるのが、そういった効果音に該当します。

・海や浜辺の波音
・シャワーの音
・火が燃えている音
・嵐の音
・キィキィと物が揺れる音
・セミなど虫の鳴き声

●再生完了まで待機する

たとえば銃撃戦のシーンなど、効果音が鳴り終わるまで進行を停止させたほうがよいケースがあります。

```
[LoadSound id='0' path='./resource/Sound/se01.mp3']
[PlaySound id='0' mode='PLAY' wait_PlayEnd='true']

効果音が鳴り終わってからこのテキストは表示されます。[p]
```

『wait_PlayEnd』パラメータは、[PlaySound]と[SetFixedSoundTrans]の両方で指定できます。

『ボイスを再生する』

●アマチュアゲームでもボイスは珍しくない

導入にはコストがかかりますが、作品を華やかにしてくれるのがキャラクターボイスです。アマチュアゲームでも商業作品並みに声優を起用しているケースも少なくない時代です。

短めのサウンドという点で、効果音と同じ感覚で考える方もいると思いますが、LemoNovel AS3 ではボイス再生用のコマンドが用意されています。

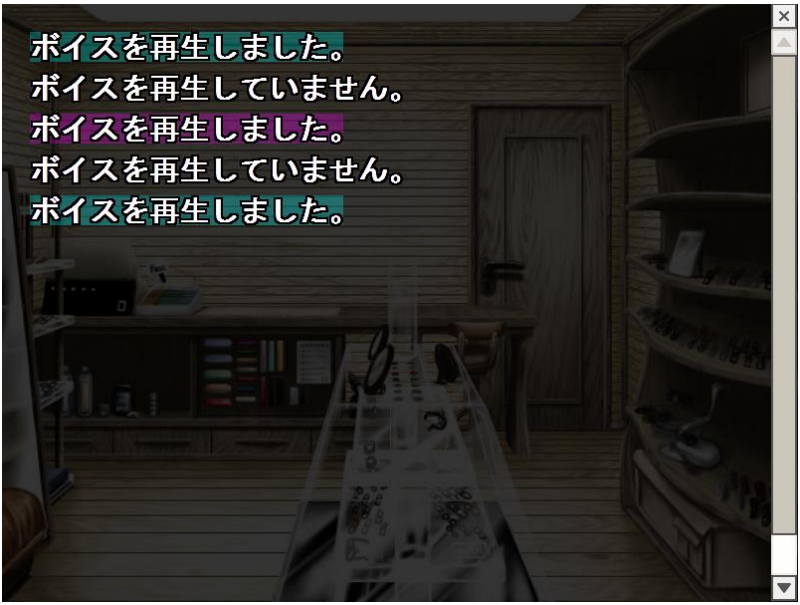
```
[Voice  path='./resource/Voice/voice01.mp3']

ボイスを再生しました。[p]

次のページに読み進めると停止します。[p]
```

この[Voice]コマンドのパラメータは『path』だけです。ボイスファイルを指定すれば、それで記述は完了します。

ボイス再生が普通の効果音と違うのは、ひとつは 1 ページにつき 1 ファイルしか使えないことです。つまり全画面で表示するノベルゲームでは、ボイスを使うのは難しいということです。現実にはボイスを使用するのは、メッセージウィンドウが画面下部に表示されるタイプがほとんどなので、たいした問題ではないと思います。

もうひとつは、次のページに進むと自動的に再生が停止することです。しかし停止条件については初期設定で変えることができます。初期設定（ini）ファイルの「ボイスの再生停止条件」という項目で、次のボイスが再生するまで停止しないという設定にすることが可能なのです。どちらがいいかは各自の好みで決めましょう。

また、[Voice]コマンドで再生したサウンドは履歴画面に登録され、何度でも聞き返すことができるという特徴があります。ボイスが割り当てられているページのテキストは、初期設定（ini）ファイルで指定した背景色で強調されるようになっています。マウスのポイント時には違うカラーにできたり、細かいカスタマイズが可能です。



『ムービーを再生する』

●作品に高級感を出すムービー

アマチュアゲームでも歌入りのオープニングムービーは珍しくありません。ムービーは特別な技術が必要なので、ほとんどが外注して制作します。それなりのコスト負担があるわけですが、作品が段違いに華やかになるので、余裕があるなら採用したいものです。

```
ムービーを再生します。[p]

[LoadMovieLv level='1' kind='FLV' path='./resource/Mov/sample.mp4']
[LoadMovieLv level='1' kind='OTHER' path='./resource/Mov/sample.swf']
```

ムービーを再生するには、[LoadMovieLv]コマンド 1 行の記述だけです。『level』はファイルを読み込む階層を指定するパラメータです。1 から 999 まで指定できますが、ムービーは普通、複数再生することはありませんので、1 で固定してよいでしょう。

『kind』パラメータはムービーの種類を指定します。『FLV』なら Flash Player で再生可能な動画で、MP4 や FLV がスタンダードです。このパラメータを省略するか『OTHER』を指定すると、SWF ファイルを扱えます。

ウェブ上の Flash ゲームを遊んだことのある人も多いと思いますが、それらは SWF 形式です。つまり SWF ファイルを埋め込むことで「ゲーム内ゲーム」のプレイも可能となります。

すなわちアクションやシューティングなど、ミニゲームでユーザーを楽しませることができるので、スキルのある人はチャレンジしてみる価値があると思います。

●ムービーを再生しながらシナリオを進める

LemoNovel AS3 でのムービーは、画面全体に表示することでしか再生できないわけではありません。画面サイズよりも小さいムービーを好きな座標に配置して、それを再生しながらシナリオを進行させることも可能です。

```
[LoadMovieLv level='1' kind='FLV' path='./resource/Mov/sample.mp4' adjustSize='false' posX='200' posY='150' wait='NONE']

ムービーが流れても。[p]
シナリオを進められます。[p]
```

画面サイズが 800×600 で、ムービーのサイズが 400×300 と仮定します。『adjustSize』はムービーのサイズを画面に合わせて伸縮するかどうかを指定するパラメータです。『false』にすると元のサイズで表示されるので、さらに画面中央に表示されるよう、座標を決定しています。

そしてムービーの再生完了を待機するかしないかを指定する『wait』パラメータを、『NONE』とします。これでムービーを再生しながら、通常どおりにメッセージやキャラクター画像の表示ができます。

『ウェイトとスキップ』

●地味だけど大切なウェイト演出

ゲームはだいたいいつも動いているものですが、それだけに一時停止することが、演出上の大事な手法になります。

```
[SetFixedBGTrans mode='FADEOUT_BLACK' time='1000']
[StartTrans]

[Wait time='2000']

[LoadBG path='./resource/BG/bg01.jpg']
[SetFixedBGTrans mode='CROSSFADE' time='1000']
[StartTrans]

背景切り替えの間に２秒ウェイトしました。
```

画面を暗くして新しい背景を表示……というスクリプトですが、間に[Wait]コマンドを挟んでいます。ここで指定した時間、進行が停止するのです。上記の例では、暗いシーンが 2 秒間続きます。こうすることで、時間経過を明確に示しているのです。他、ほんのちょっとしたシーンでウェイトを使えば、絶妙な演出や余韻になるでしょう。

LemoNovel AS3 におけるウェイト機能は、デフォルトではユーザー操作でスキップできます。つまりウェイトの途中でマウスクリックしたりエンターキーを押したりすると、ウェイトが中断されて進行再開するのですが、せっかくの演出や余韻がもったいない……ということになりかねません。そこでパラメータを一工夫します。

```
[Wait time='2000' skip='false']

ウェイトのスキップはできませんでした。[p]
```

『skip』はユーザー操作でウェイトをスキップできるかのパラメータです。ここではスキップさせたくないな、という場合には『false』を指定しましょう。これで何があっても 2 秒間ウェイトすることになります。

逆に、ユーザーが操作しなければ進行を再開しないという状況にもできます。

```
[Wait click='true']

マウスクリックでウェイト解除しました。[p]
```

『time』パラメータを省略し、代わりに『click』パラメータを記述しています。これに『true』を指定すると、マウスクリックやエンターキー押下をしない限り、先に進めないようになります。あまり多用するとユーザーにとっては不便なので、使いどころはしっかり考えましょう。

●パラメータでのウェイトとスキップ

効果音の項で「鳴り終わるまで先に進まない」という機能を紹介しました。これもウェイトなわけですが、画像表示の際にも、ウェイトするかしないかを調整することができます。

```
[LoadChar id='0' path='./resource/Char/01.png']
[SetFixedCharTrans id='0' mode='FADEIN' time='500' wait='false']
[StartTrans]

キャラクターが表示しきらないうちから、このテキストが流れます。[p]
```

『wait』パラメータに『false』を指定すれば、エフェクトの完了を待たないで先に進みます。こうしておけばキャラクター画像のエフェクトとテキストが流れるのを、同期させることができるのです。

また、エフェクトをスキップさせない指定が可能です。

```
[LoadBG path='./resource/BG/bg01.jpg']
[SetFixedBGTrans mode='CROSSFADE' time='1000' skip='false']
[StartTrans]

クロスフェードのエフェクトをスキップできませんでした。[p]
```

やはり『skip』パラメータに『false』と指定すれば、ユーザー操作によってエフェクトがスキップされることはなくなります。オープニングやエンディングシーンなど、ありのままを見せたい時には、スキップできないようにするとよいでしょう。

『画面/端末を揺らす』

●演出に欠かせない画面揺らし

何かにぶつかる、激しくバトルする、地震が起こる……そんなシーンで画面揺らしの演出が大活躍します。これには[Quake]コマンドを使用します。

```
[Quake obj='ALL' time='1000' hMaxAmp='50' vMaxAmp='50']
[StartQuake]

画面全体が１秒間揺れました。[p]
```

まずは揺らす対象を『obj』パラメータで指定するのですが、この例では『ALL』つまり画面全体が対象です。省略しても『ALL』になります。もっともスタンダードな使い方になるでしょう。他に指定できるのは『BG』が背景レイヤー、『CHAR』がキャラクターレイヤー、『MSG』がメッセージレイヤーです。

最大振幅を示すのが『hMaxAmp』と『vMaxAmp』の両パラメータです。前者が水平方向、後者が垂直方向になります。あまり値を大きくすると、画面を見る人は揺れすぎで気持ち悪くなるので気をつけてください。

そして画像エフェクトのときのように、[StartQuake]コマンドで画面揺らしをスタートします。このコマンドにも『wait』と『skip』のパラメータがあるので、状況によって振動の完了を待機するかしないか、スキップさせるかさせないかを決められます。

```
[Quake obj='BG' time='2000' hMaxAmp='50' vMaxAmp='50']
[StartQuake wait='false']

振動の完了を待機しません。[p]
```

```
[Quake obj='BG' time='2000' hMaxAmp='50' vMaxAmp='50']
[StartQuake skip='false']

振動をスキップできませんでした。[p]
```

●様々な揺らし方

先程の例では『time』パラメータで振動時間を指定しましたが、これを省略することでずっと揺らし続けることができます。

```
[Quake obj='ALL' hMaxAmp='10' vMaxAmp='10']
[StartQuake wait='false']

細かく揺れ続けています。[p]

[EndQuake]

振動を完了しました。[p]
```

振動を完了させるには[EndQuake]コマンドを使用します。これはパラメータがないので、そのまま記述してください。

また、キャラクターを複数表示しているシーンで、誰かひとりだけを揺らすことができます。

```
[LoadChar id='0' path='./resource/Char/01.png']
[LoadChar id='1' path='./resource/Char/02.png']
[SetFixedCharTrans id='0' mode='FADEIN' time='500' posX='0' posY='0']
[SetFixedCharTrans id='1' mode='FADEIN' time='500' posX='400' posY='
0']
[StartTrans]
```

```
[Quake obj='CHAR' id='1' time='1000' hMaxAmp='10' vMaxAmp='10']
[StartQuake]

ＩＤ１だけが揺れました。[p]
```

表示させたふたりのキャラクターのうち、『id』パラメータで片方だけを指定しています。この指定は『id='1,2'』のようにコンマで区切って複数指定することも可能です。

そして揺らし方の種類も変更できます。

```
[Quake type='SHAKE' interval='1000' hMaxAmp='10' vMaxAmp='10']
[StartQuake]

画面全体がシェイクで揺れています。[p]
```

新しく『type』パラメータを記述しています。これまでこのパラメータは省略していて、その場合には『QUAKE』という指定がされていたのですが、『SHAKE』とするとドリンクをシェイクするように、一定方向にのみ揺れるのです。

『interval』というパラメータは、振動させる周期に対して時間指定します。この例では画面が揺れ始めてから元の位置に戻るまで１秒かかり、かなりゆったりした揺れ方ということになります。

●Android 限定のバイブレーション機能

AIR Mobile 版の Android 限定になりますが、端末をバイブレーションさせることができます。いきなりこれが来ると驚くので、ホラー作品などでおおいに活躍するでしょう。

```
[VibeCtrl type='TIME' time='1000']
```

[VibeCtrl]コマンドは 2 つのパラメータが必須です。『type』は『TIME』を指定すると、『time』で指定した時間バイブレーションします。

また、『PATTERN』と指定すると振動パターンを使うことができます。

```
[VibeCtrl type='PATTERN' time='500,1000,2000,1000' repeatIdx='0']
```

『repeatIdx』というパラメータは、バイブレーションを繰り返したいときに使用します。繰り返す際のインデックス（先頭は 0）を指定するのです。停止と振動の繰り返しなので、この例では 0.5 秒停止、1 秒振動、2 秒停止、1 秒振動というパターンになります。

```
[VibeCtrl type='STOP']
```

繰り返しているといつまでも振動が終わらないので、このように停止したい時点で『type=STOP』と記述します。

『選択肢とジャンプ』

●ゲーム性を高める選択肢

ノベル・アドベンチャーゲームに欠かせないのが、シナリオを分岐させる選択肢です。これを使いこなせるようになれば、LemoNovel AS3 での制作はいっそう楽しくなります。

選択肢の基本としては、「選択肢コマンド」「選択先となるラベル」「再び一本道に戻るためのジャンプコマンド」を記述します。

```
これから選択肢を出します。[p]

[Link label='page1']選択肢１[/Link][r]
[Link label='page2']選択肢２[/Link]
[StartSelect]

*page1
[ClearMsg]
選択肢１を選びました。[p]
[Goto label='gouryu']

*page2
[ClearMsg]
選択肢２を選びました。[p]
[Goto label='gouryu']

*gouryu
合流しました。[p]
```

まず[Link]コマンドです。『label』パラメータにラベル名を記述しています。ラベルとはシナリオ上の目印のことで、アスタリスク（*）に任意の文字列を加えたものです。『label』パラメータに記述する際には、アスタリスクは必要ありません。

次に[/Link]で閉じることでひとつの選択肢が完成します。そして[StartSelect]コマンドによって選択肢表示され、同時に進行停止します。選択肢を選ばない限り、先に進めないようになります。

選択肢を選んだら、指定したラベルのテキストが表示されるはずです。ラベルの直後に[ClearMsg]と記述していますが、これは画面上のテキストを消去するコマンドです。選択肢を選んでも[p]コマンドのように消去されないため、このコマンドが必要になるのです。最後に[Goto]コマンドの『label』パラメータに、同じようにラベルを指定すれば、分岐したシナリオが一本道に戻ります。

●異なるファイルの間でジャンプ

実際にゲーム制作をする場合は、複数のシナリオファイルに分けるのが管理の都合上好ましいですが、[Link]と[Goto]の両方で、現在のファイルと異なるファイルを指定してジャンプすることができます。

```
現在のファイルは 01.txt です。[p]

[Link  path='./script/02.txt']選択肢 1 [/Link][r]
[Link  path='./script/03.txt']選択肢 2 [/Link]
[StartSelect]
```

『path』パラメータにシナリオファイルを指定します。これで 01.txt のシナリオは終わっていて、選択肢によって新しいファイルを読み込ませるわけです。なお、『path』パラメータを使用していて『label』パラメータを省略する場合は、シナリオファイルの最初から読み込まれます。

```
[ClearMsg]
02.txt にジャンプしました。[p]
[Goto  path='./script/04.txt']
```

※選択肢１を選んだ場合

```
[ClearMsg]
03.txt にジャンプしました。[p]
[Goto  path='./script/04.txt']
```

※選択肢２を選んだ場合

```
04.txt にジャンプしました。[p]
```

※各選択肢から一本道に戻ってきた

この方法だと選択肢が多くなるほどファイル数も増えますが、それでもひとつのファイルに延々と記述していくよりは、わかりやすくていいと思います。

●リンクテキストの装飾

選択肢もメッセージレイヤーに表示するので､通常のテキストと同じように､あるいは選択肢ならではの装飾をすることができます。

パラメータ名	概要
linkColor	非ポイント時のフォントカラー
linkColor_P	ポイント時のフォントカラー
linkEdgeColor	非ポイント時の袋文字の縁のカラー
linkEdgeColor_P	ポイント時の袋文字の縁のカラー
linkUnderline	非ポイント時のアンダーライン表示
linkUnderline_P	ポイント時のアンダーライン表示
sound_In	ポイント時に再生するサウンドファイル
sound_Out	ポイント解除時に再生するサウンドファイル
sound_Click	クリック時に再生するサウンドファイル

記述例は以下のようになります。

```
[Link label='page1' linkColor_P='0xFFFF00']黄選択[/Link][r]
[Link label='page2' sound_In='./resource/Sound/se01.mp3']音選択[/Link]
[StartSelect]
```

特によく使うと思われるのがこの 2 種類で、『linkColor_P』パラメータはマウスカーソルを選択肢にポイントしたときのカラーです。この例では黄色に変化します。『sound_In』パラメータは、同じくポイントした時に再生されるサウンドファイルです。

たとえばホラーゲームなら赤いテキストに変化させるとか、不気味な音声を出すとか、様々な工夫が考えられると思います。また、[Link]コマンドは[Font]コマンドの持つすべてのパラメータを使用できますので、それで装飾するのもいいでしょう。

●別のメッセージレイヤーで選択肢表示

アドベンチャータイプのゲームだと、通常のテキストは画面下部に表示するが、選択肢だけは画面中央に表示させるというケースがあります。

これを実現するには……。

```
選択肢を表示します。[p clear='false']

[AddObj type='MSG' id='1']
[ActiveMsgLayer id='1']
[MsgLayer id='1' visible='true' posX='150' posY='150' width='500' height
='110']
[Link label='page1']選択肢 1 [/Link][r]
[Link label='page2']選択肢 2 [/Link]
[StartSelect]
```

```
*page1
[ClearAllMsg]
[DelObj type='MSG' id='1']
選択肢１を選びました。[p]
[Goto label='gouryu']

*page2
[ClearAllMsg]
[DelObj type='MSG' id='1']
選択肢２を選びました。[p]
[Goto label='gouryu']

*gouryu
合流しました。[p]
```

この例が示すのは、メッセージレイヤーをもうひとつ生成して、そこで選択肢の処理をするというプロセスです。

最初の「選択肢を表示します。」で[p]コマンドに『clear』パラメータというのがありますが、これを『false』にすると、クリック待機状態が解除されたあともテキストがクリアされません。つまり次のページに行かないという処理になるのです。

テキストが残っている状態で、メッセージレイヤーを新たに生成します。キャラクター画像の項で紹介した[AddObj]コマンドが再登場です。ここで『type='MSG' id='1'』と指定するのです。ただしメッセージレイヤーは、生成しただけでは使えません。[ActiveMsgLayer]コマンドで、そのレイヤーを操作対象とする必要があるのです。さらに[MsgLayer]コマンドで、座標や大きさを指定します。『visible』パラメータを『true』にするのを忘れないでください。

そうしたら通常どおりに選択肢の記述をします。画面のように、新しく生成したほうのメッセージレイヤーに表示されました。

各選択肢を選んだあとの記述も、通常とは少し違います。[ClearAllMsg]は、すべてのメッセージレイヤーをクリアするコマンドです。そして[DelObj]は、指定したレイヤーを消去するコマンドです。選択肢が終われば、画面中央のメッセージレイヤーは必要なくなるわけですから、これで消去しておくのです。

●サブルーチン機能

プログラム用語で、サブルーチンというものがあります。

これは、繰り返し使用することが前提の処理をひとまとめにしたものです。これを呼び出すのも一種のジャンプと言えます。

ゲーム制作ツールにおいてもスタンダードな機能ですが、LemoNovel AS3では[Gosub]というコマンドを使用します。まずは以下のように記述します。

```
サブルーチンを呼び出します。[p]

[Gosub path='./script/sub.txt']
```

『path』パラメータに指定している sub.txt には、以下のように記述します。

```
[AddObj type='MSG' id='1']
[ActiveMsgLayer id='1']
[MsgLayer id='1' visible='true' color_BG='0x000000' posX='0' posY='0' wi
dth=#System.width height=#System.height alpha='1']

サブルーチンです。[p]

[DelObj type='MSG' id='1']
[ActiveMsgLayer id='0']
[Return]
```

サブルーチン内で行っているのは、やはり新しいメッセージレイヤーを生成していることです。

ここで生成しているメッセージレイヤーは、画面が真っ黒で完全に不透明です。『width』パラメータの『#System.width』と『height』パラメータの『#System.height』は、画面のサイズが何であろうと、その最大横幅と最大縦幅を指定する特殊な記述方法です。横 800×縦 600 であれば、『width』に 800 が、『height』に 600 が指定されます。

そして必要がなくなったらメッセージレイヤーを消去し、[Return]コマンドで元の画面に復帰するのです。実際にサブルーチンを使う際は、もちろんこんなシンプルな記述では済みませんが、基本はこうなっているということで覚えてください。

『変数と条件分岐』

●あらゆるゲームに欠かせない変数

選択肢がノベル・アドベンチャーゲームに欠かせない要素なら、変数はそもそもゲームにとって欠かせない要素です。「王様の話を聞いたら冒険に出られるようになった」などのイベントは、変数によって管理されるのです。

変数は読んで字のごとく、状況に応じて値が変わる数です。ただしゲームにおいては数字以外の文字列なども入ったりします。いろいろ説明するよりも、実例を見ていただきましょう。

```
[Var test='10']
[Var name='山田太郎']

『test』という名前の変数に 10 を代入しました。[r]
『name』という名前の変数に「山田太郎」を代入しました。[p]
```

変数を使用するには[Var]というコマンドを記述します。変数名はアスタリスク (*) やピリオド (.) が含まれていなければ自由で、この例では『test』『name』が変数名です。『test』には「10」という数字、『name』には「山田太郎」という文字列を入れています。

変数は値を入れただけでは何の役にも立ちませんので、画面上に表示させてみましょう。これを「変数を参照する」という言い方をします。

```
先程の変数の値を表示します。[r]
[r]
[Output msg="@user.test"][r]
[Output msg="@user.name"][p]
```

[Output]はメッセージ出力用のコマンドで、『msg』に指定した値が表示されます。これに『@user.test』『@user.name』が指定されていますが、生成した変数の値を実際に表示するには『@user.変数名』という書式を用いるのです。

ここでダブルクォーテーションで囲っていることに注意してください。シングルクォーテーションだと「@user.test」「@user.name」という文字列自体が表示されてしまいます。パラメータの値に変数を指定する際はダブルクォーテーションと覚えてください。

とにかくこれで「10」と「山田太郎」が表示されますが、試しに他の値に書き換えてみてください。その値が表示されるはずです。

●ユーザー変数とシステム変数

LemoNovel AS3 で使用する変数は 2 種類あり、ユーザー変数とシステム変数といいます。ユーザー変数はゲームデータに保存されますが、システム変数はシステムに保存されます。先程の例ではユーザー変数を使っていました。

恋愛ゲームで言えば、女の子の好感度はユーザー変数、クリア後に出てくるおまけモードはシステム変数に保存します。これは厳密に区別する必要があり、たとえばおまけモードをユーザー変数で管理してしまうと、システムに保存されないので、終了して再起動したら、タイトル画面から消えてしまう……ということになってしまうのです。

おまけモードの作り方については第 4 章で取り上げるとして、システム変数の基本的な使い方を説明します。

```
[SysVar test='100']
[SysVar name='日本']

代入したシステム変数を表示します。[r]
[r]
[Output msg="@sys.test"][r]
[Output msg="@sys.name"][p]
```

[Var]が[SysVar]に変わり、[Output]で指定している値が『@sys.変数名』に変わっています。違いはこれだけですので、あまり難しくありません。

●変数を計算する

変数は単純に値を代入するだけでなく、計算することができます。攻略対象の好感度をイベントによって上下させるのは、この計算によるものなのです。

```
[Var hanako='5']

花子の好感度を 5 に設定しました。[p]

[Link label='page1']Ａ：花子を褒める[/Link][r]
[Link label='page2']Ｂ：花子を罵る[/Link]
[StartSelect]

*page1
[ClearMsg]
[Var hanako="@user.hanako + 1"]
花子「ありがとう！」[p]
[Goto label='gouryu']

*page2
[ClearMsg]
[Var hanako="@user.hanako - 1"]
花子「ひどいわ！」[p]
[Goto label='gouryu']

*gouryu
花子の好感度が[Output msg="@user.hanako"]になりました。[p]
```

最初にユーザー変数「hanako」を 5 として、選択肢を選ばせます。褒めると花子が喜んで好感度が上がりますが、罵ると怒って好感度が下がる……という流れです。

選択した先で、計算した結果の値を新たに代入していることに注目してください。『@user.変数名』に足し算や引き算をする、これが LemoNovel AS3 における変数計算の書式です。選択肢 A を選ぶと好感度は 6 に、B を選ぶと 4 になります。

なお、こういった計算の際には、その式を必ずダブルクォーテーション（"）で囲まなければいけません。シングルクォーテーション（'）で囲むと、その中身が文字列と判断されてしまいます。たとえば褒めるを選択すると、「花子の好感度が@user.hanako + 1 になりました」と表示されてしまうのです。

計算に使う記号を演算子といいますが、LemoNovel AS3 の四則演算は次の演算子を使用します。

足し算	引き算	掛け算	割り算
+	-	*	/

掛け算は「×」ではなく、割り算は「÷」ではありませんので注意しましょう。そして文字列に関しては「+」で連結することができます。

```
[Var  moji1='関東地方']
[Var  moji2='東京都']

文字を連結します。[p]

[Var  new_moji="@user.moji1  +  @user.moji2"]

[Output  msg="@user.new_moji"][p]
```

最後に「関東地方東京都」と表示されます。

●条件分岐

ただ値を代入したり計算したりするだけでは、変数をフル活用することはできません。「その時点の値によりシナリオを分岐させる」というのが、変数の真の使い方なのです。これを条件分岐といいます。

先程の好感度の例を、ちょっと変化させてみましょう。合流以降に注目してください。

```
[Var hanako='5']

花子の好感度を 5 に設定しました。[p]

[Link label='page1']Ａ：花子を褒める[/Link][r]
[Link label='page2']Ｂ：花子を罵る[/Link][r]
[Link label='page3']Ｃ：何も言わない[/Link]
[StartSelect]

*page1
[ClearMsg]
[Var hanako="@user.hanako + 1"]
花子「ありがとう！」[p]
[Goto label='gouryu']

*page2
[ClearMsg]
[Var hanako="@user.hanako - 1"]
花子「ひどいわ！」[p]
[Goto label='gouryu']
```

```
*page3
[ClearMsg]
花子「どうしたの？」[p]
[Goto label='gouryu']

*gouryu
[If exp="@user.hanako == 6"]
好感度がプラスされました。[p]
[ElseIf exp="@user.hanako == 4"]
好感度がマイナスされました。[p]
[Else]
好感度に変化はありません。[p]
[EndIf]
```

条件分岐の基本となるのが、[If]～[ElseIf]～[Else]～[EndIf]の一連のコマンドです。まず[If]で最初の条件分岐判定を行います。『exp』パラメータに『@user.hanako == 6』と記述していますが、これはユーザー変数「hanako」が 6 だったらこの処理になる、という意味になります。なお、条件に一致することを「真である」という言い方をします。

もし条件に一致しなかったら、[ElseIf]で再び条件分岐判定をします。ここでは『@user.hanako == 4』ですが、やはりユーザー変数「hanako」が 4 だったらこの処理になる、という意味です。

いずれも条件に一致しなかった(この例の場合は選択肢 C を選んだ)場合は、[Else]での処理が行われます。そして最後に[EndIf]を記述して、条件分岐を終了させます。

このうち[ElseIf]と[Else]については必須ではありません。しかし[If]と[EndIf]については、必ず対で記述する必要があります。

さて、「==」という演算子が出てきました。ゲーム制作においては、普通の数学では見られない、条件分岐のための演算子を覚える必要があります。

演算子	意味
==	両辺が等しい
!=	両辺が等しくない
<	右辺が大きい
<=	右辺が大きいか等しい
>	左辺が大きい
>=	左辺が大きいか等しい
&&	「A && B」とした時、A も B も真なら処理を実行
||	「A || B」とした時、A または B が真なら処理を実行

たとえば『exp="@user.hanako != 10"』であれば、ユーザー変数「hanako」が 10 でないのならこの処理になる、となります。「&&」と「||」については、実例を見てみましょう。

```
// [Var hanako='1']
// [Var youko='1']

[If exp="@user.hanako == 1 && @user.youko == 1"]
花子と洋子の好感度は、どちらも 1 です。[p]
[ElseIf exp="@user.hanako == 1 || @user.youko == 1"]
花子と洋子の好感度は、どちらかが 1 です。[p]
[Else]
花子と洋子の好感度は、どちらも設定されていません。[p]
[EndIf]
```

最初のコメントアウトされている部分を両方、またはどちらか削除してからこのスクリプトを試してみると、誰の好感度が設定されているかによってテキストが変わってくるのです。ゲームでは複数の条件を比較するシーンがよくあるので、そのときはこのように記述してください。

●もうひとつの条件分岐方法

条件分岐は基本的に[If]～[EndIf]で行いますが、もうひとつの方法として、コマンドのパラメータに条件式を記述することができます。

```
// [Var abc='1']

次のテキストはユーザー変数「abc」が 1 ならば表示されます。[p]

[Output msg='表示されます。' __cond="@user.abc == 1"][p]
```

こちらも最初のコメントアウトされている部分を削除、またはそのままにして試してみてください。『__cond』は一部を除くほとんどのコマンドで使用できるパラメータです。これに条件式を指定し、それが真ならばコマンドが実行されます。単一のコマンドを条件分岐で処理したい場合は、[If]～[EndIf]よりもコンパクトに記述できるのでおすすめです。

『ボタンで選択する』

●作品をスタイリッシュにするボタン

ほとんどのゲームは、システムデザインなどに作品に合った雰囲気のボタンを使用しています。中世が舞台ならファンタジック、歴史ものなら和風といった具合に。ボタン機能をまったく使用しなくてもゲームにはなりますが、やはり使えた方がずっといいと思います。

ボタン画像は通常の画像とは違って、同じサイズの異なる画像を複数連結したものを使います。いくつ連結するかは、用途によって違ってきます。サンプルのボタン画像を見てみましょう。

3 つ連結されていますが、左から「カーソルがポイントされていない」「クリック状態」「カーソルがポイントされている」となります。この画像をボタンとして使う場合は、3 つのいずれかが画面に表示されるのです。

また、ボタンには「無効状態」を用意したい場合があります。

4 つめに「無効状態」の画像を連結しておけば、そのように使うことができます。どちらを使うかは、パラメータによって決まります。

●選択型ボタン

ボタンには「選択型」「常駐型」の 2 種類があります。選択型は読んで字のごとく、選択肢として用います。常駐型もまたそのままの意味で、メッセージウィンドウに常に付属するシステムボタンなどに用います。

まずは選択型ボタンの生成から見ていきましょう。記述するのは[Button]というコマンドです。

```
[Button id='SEL1' group='SELECT' dstLayer='MSG' posX='300' posY='200' path_Pic='./resource/Button/button1.jpg']
[Button id='SEL2' group='SELECT' dstLayer='MSG' posX='300' posY='250' path_Pic='./resource/Button/button2.jpg']
[Button id='SEL3' group='SELECT' dstLayer='MSG' posX='300' posY='300' path_Pic='./resource/Button/button3.jpg']
[StartSelectBtn group='SELECT' var='selId']
```

ひとつひとつパラメータを見ていきます。『id』はこれまでのコマンドと同様、そのボタンに割り当てられる ID です。

『group』はそのとき画面に表示するボタンを、グループとして管理するために指定します。この例では 3 つともが「SELECT」グループのボタンである、という意味になります。

『dstLayer』は、そのボタンをどのレイヤーに埋め込むかを指定します。『MSG』とすればメッセージレイヤーに埋め込むことになり、メッセージレイヤーを消去する操作をすれば、同時にボタンも消去されることになるのです。

そして座標と画像パスを指定するのですが、最後に[StartSelectBtn]コマンドを記述しなければボタンは機能しません。ここで『group』に各ボタンに指定したのと同じグループ名を指定します。『var』は変数名の指定で、クリックしたボタンに応じて『id』の値が格納されます。つまり 1 つめのボタンをクリックすれば『selId=SEL1』となるのです。

選択型ボタンの用途は、先に述べたように選択肢です。ボタンをクリックして格納された変数の値によって、ジャンプ先を変える……というのが基本的な使い方になります。このことを踏まえ、[StartSelectBtn]の直後にスクリプトを加筆してみましょう。

```
[Goto label='page1' __cond="@user.selId == SEL1"]
[Goto label='page2' __cond="@user.selId == SEL2"]
[Goto label='page3' __cond="@user.selId == SEL3"]
```

『id』に『SEL1』を指定しているボタン 1 をクリックすると、「page1」にジャンプするわけです。以下同様に、ボタン 2 なら「page2」へ、ボタン 3 なら「page3」へジャンプします。

なお、選択型のボタンは用が済めば、つまり先に進もうとボタンをクリックした時点で消去されます。もしあえて消去したくないなら……。

```
[StartSelectBtn group='SELECT' var='selId' delBtn='false']
```

『delBtn』パラメータを『false』と指定することで、そのまま表示させておくことができます。しかし結局は、どこかのタイミングでボタンを消去しなければなりません。

```
[DelButton id='SEL1,SEL2']
[DelButton group='SELECT']
```

『DelButton』コマンドの『id』パラメータで各 ID を、または『group』パラメータでグループ名を指定します。『group』パラメータで指定して一気に消去することがほとんどだと思いますが、個別に消去したい場合は『id』パラメータを使用しましょう。コンマで区切って任意の ID を選ぶことができます。

●常駐型ボタン

常駐型ボタンの生成は、次のようになります。

```
[Button id='SEL1' type='FIXED' dstLayer='MSG' posX='300' posY='200'
path_Pic='./resource/Button/button1.jpg']
[Button id='SEL2' type='FIXED' dstLayer='MSG' posX='300' posY='250'
path_Pic='./resource/Button/button2.jpg' with_Invalid='true']
[Button id='SEL3' type='FIXED' dstLayer='MSG' posX='300' posY='300'
path_Pic='./resource/Button/button3.jpg']
```

選択型と違うのは、ひとつは『type』というパラメータがあることです。これは省略すると自動的に選択型になるのですが、『FIXED』と指定すれば常駐型となります。もうひとつは[StartSelectBtn]コマンドが必要ないことです。

さて、2 つめのボタンには『with_Invalid』というパラメータを設けています。『true』だと先に触れた 4 つ連結された画像を使えるようになります。常駐型ボタンは一度押下すると無効化されるのですが、実際に上記のスクリプトを実行して、無効化されると同時に 4 つめの画像が表示されることを確認してください。

無効化されたままでは使えませんので、再び押下できるようにしなければなりません。

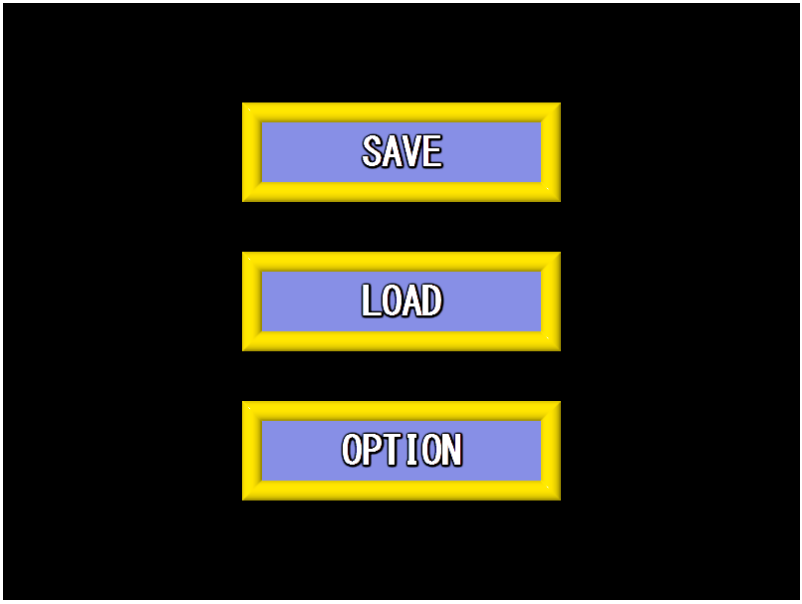
```
[ChgButton id='SEL1' enabled='true']
```

ボタンの状態を変化させる[ChgButton]コマンドで、ボタンの有効・無効を指定する『enabled』パラメータを『true』にします。常駐型ボタンは、有効と無効の切り替えをしっかり意識して使用する必要があるのです。

●ボタンにキャプションを用いる

グラフィックソフトでボタンを作成する際には、通常は文字も含めてデザインしますが、枠だけのボタンを使用し、文字は LemoNovel AS3 のシステムで表示させるという方法があります。これをキャプション機能といいます。

```
[Button id='SAVE' type='FIXED' dstLayer='MSG' caption='SAVE' posX='240' posY='100' font_Size='40' path_Pic='./resource/Button/button.png']
[Button id='LOAD' type='FIXED' dstLayer='MSG' caption='LOAD' posX='240' posY='250' font_Size='40' path_Pic='./resource/Button/button.png']
[Button id='OPTION' type='FIXED' dstLayer='MSG' caption='OPTION' posX='240' posY='400' font_Size='40' path_Pic='./resource/Button/button.png']
```



ボタンにキャプションを表示する『caption』パラメータに、好きな内容を指定します。画像パスに指定しているのは、すべて同じ画像ということに注目してください。この機能を使えば、いくつものボタンを作らなくていいというメリットがあります（ただしデザイン面では妥協するということになります)。

また、キャプションはフォントサイズやカラーなど、様々な装飾指定ができます。下に挙げる他にもいくつものパラメータがありますので、タグリファレンスで確かめてみてください。

パラメータ名	概要
font_Size	キャプションのフォントサイズ
font_Color	通常状態でのキャプションのフォントカラー
font_Color_Point	ポイント状態でのキャプションのフォントカラー
font_Color_Press	押下状態でのキャプションのフォントカラー
font_Color_Invalid	無効状態でのキャプションのフォントカラー
align	キャプションの水平方向の表示位置揃え

●キャプションだけのボタン

実はボタンは､必ず画像を用意しなければならないということはありません。キャプションを指定して画像パスを省略すれば、それが自動的にボタンになります。

```
[Button id='SAVE' type='FIXED' dstLayer='MSG' caption='SAVE' posX='300' posY='200']
```

この方法で作成すると、キャプションの文字列の縦幅と横幅がそのままボタンの大きさになります。

●ボタンのテンプレートを作成する

キャプションの装飾指定について､「[Button]コマンドのひとつひとつに、それらを指定しなければならないのか？」と思った方がいると思います。

そのような煩雑な記述を避けるために、ボタンのテンプレート機能があります。

```
[RegistBtnTemp name='SELBTN' path_Pic='./resource/Button/button1.jpg' font_size='30' font_Color='0xFFFFFF' font_Color_Point='0x0000FF' font_Color_Press='0x0000FF' font_Color_Invalid='0x888888' sound_In='./resource/Sound/point.mp3' sound_Click='./resource/Sound/click.mp3']
```

一行がかなり長くなりましたが、[RegistBtnTemp]コマンドによりテンプレートを作成します。フォントのサイズやカラー、ポイント時のカラーやサウンドなどを指定しています。そして『name』パラメータにテンプレート名を指定しています。

このテンプレートを使用してボタンを表示してみます。

```
[Button id='SEL1' group='SELECT' dstLayer='MSG' caption='選択肢Ａ' p
osX='300' posY='200' template='SELBTN']
[Button id='SEL2' group='SELECT' dstLayer='MSG' caption='選択肢Ｂ' p
osX='300' posY='250' template='SELBTN']
[Button id='SEL3' group='SELECT' dstLayer='MSG' caption='選択肢Ｃ' p
osX='300' posY='300' template='SELBTN']
[StartSelectBtn group='SELECT' var='selId']
```

『template』にテンプレート名を指定すれば、[RegistBtnTemp]コマンドで指定したパラメータがすべて適用されます。特にキャプションを使用するなら、テンプレート機能は必要不可欠でしょう。

『ヒントを表示する』

●リンクやボタンを補助する

ヒントはリンクやボタンにポイントを合わせたとき、任意のテキストなどを表示する機能です。たとえば「SAVE」というボタンに「セーブします」といった解説を表示させたりできます。

ヒントはテンプレートを作成することからはじめます。ボタンのテンプレートと理屈はまったく同じです。

```
[RegistHintTemp name='TITLE_MENU' posType='OBJ' location='RB' font_size='16' fon_Color='0xFFFFFF' path_BG='./resource/System/hintBG.png']
```

［RegistHintTemp］コマンドで、さまざまなパラメータを指定しています。『posType』はヒントの表示位置の基準です。このパラメータの値は 4 種類あります。

値の種類	概要
OBJ	表示対象のオブジェクト
POINTER	ポインタ
POINTER_FOLLOW	ポインタ（表示後も追従）
FIX	固定座標

次に『location』はヒントの配置に関するパラメータです。『posType』と合わせると、この例の場合は「リンク項目の右下（Right、Bottom）に表示」を意味するのです。『location』のパラメータの値は 20 種類以上もあるのですが、公式サイトのタグリファレンス「配置の指定について」というページに載っているのでチェックしてください。

では、テンプレートを使ってヒントを表示してみましょう。

```
[Link hint='初めからプレイします。' hintTemp='TITLE_MENU']START[/Link][r]
[Link hint='続きからプレイします。' hintTemp='TITLE_MENU']LOAD[/Link][r]
[Link hint='システムを変更します。' hintTemp='TITLE_MENU']CONFIG[/Link]
```

『hintTemp』パラメータにテンプレート名を指定します。テンプレート名はメッセージレイヤーであらかじめ指定しておくこともできます。

```
[MsgLayer hintTemp='TITLE_MENU']

[Link hint='初めからプレイします。']START[/Link][r]
[Link hint='続きからプレイします。']LOAD[/Link][r]
[Link hint='システムを変更します。']CONFIG[/Link]
[StartSelect]
```

いちいち[Link]コマンドで指定しなくてもよくなりました。

また、各項目で文字色などを部分的に変更したい場合は『hintParam』というパラメータを使用します。

```
[MsgLayer hintTemp='TITLE_MENU']

[Link hint='初めからプレイします。' hintParam='font_Color=0xFF0000']START[/Link][r]
[Link hint='続きからプレイします。' hintParam='font_Color=0xFFFF00']LOAD[/Link][r]
[Link hint='システムを変更します。' hintParam='font_Color=0x0000FF']CONFIG[/Link]
[StartSelect]
```

●ボタンにヒントを使用する場合

ボタンにおけるヒントは、通常のリンクとさして変わりません。ただし、ひとつだけ注意する点があります。

```
[Button id='BTN1' group='TITLE' posX='300' posY='300' caption='START' template='SELBTN' hint='初めからプレイします。' hintTemp='TITLE_MENU']
[Button id='BTN2' group='TITLE' posX='300' posY='400' caption='LOAD' template='SELBTN' hint='続きからプレイします。' hintTemp='TITLE_MENU']
[Button id='BTN3' group='TITLE' posX='300' posY='500' caption='CONFIG' template='SELBTN' hint='ゲームを終了します。' hintTemp='TITLE_MENU' hintParam='font_Color=0xFF0000']

[StartSelectBtn group='TITLE' var='selId']
```

ボタンのテンプレートとヒントのテンプレートを併用しているのですが、『hintTemp』や『hintParam』は必ず『template』の右側に記述しなければなりません。

『画像に変化を加える』

●画像を移動する

ノベル・アドベンチャーゲームにおける重要な演出のひとつが画像の変化です。まずはもっとも基本となる画像の座標移動を見てみましょう。

```
[CharLayer id='0' posX='400' posY='100']

X 座標 400、Y 座標 100 に移動しました。[p]
```

単純に座標を変えるだけならこれで充分ですが、時間をかけて移動させたい場合には、X 座標と Y 座標の終了値を示す『dst_posX』と『dst_posY』を用います。

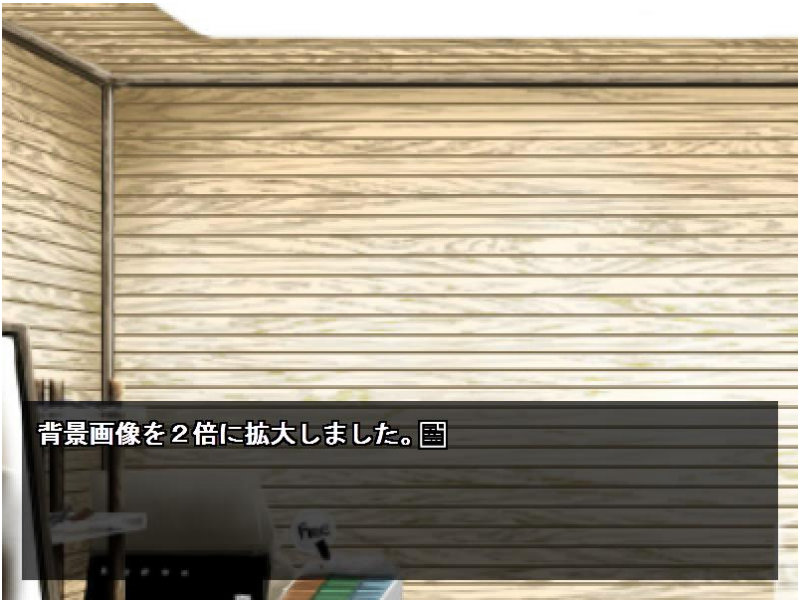
```
[SetCharTrans id='0' dst_posX='400' dst_posY='100' time='500']
[StartTrans]

X 座標 400、Y 座標 100 に 0.5 秒で移動しました。[p]
```

[SetCharTrans]は[SetFixedCharTrans]と似ていますが、キャラクターレイヤーの表示・消去以外で何かしらの変化させたいときに使うと覚えてください。

●画像の倍率を変える

何かに接近する、といったシーンで有効なのが画像の倍率の変更です。

```
[BGLayer surface='PRIMARY' scaleX='2' scaleY='2']

背景画像を２倍に拡大しました。[p]
```



『scaleX』が横の倍率、『scaleY』が縦の倍率です。値が 0～1 の間なら縮小されます。

単に倍率を変えるだけならこれで充分ですが、時間をかけて変えたいなら次のようにします。

```
[SetBGTrans dst_scaleX='2' dst_scaleY='2' time='500']
[StartTrans]

背景画像を 0.5 秒で 2 倍に拡大しました。[p]
```

背景レイヤーを変化させる[SetBGTrans]コマンドです。『dst_scaleX』が横の倍率の終了値、『dst_scaleY』が縦の倍率の終了値となります。同様のパラメータ指定がキャラクターレイヤーでもできますので試してみましょう。

座標の終了値をマイナスに指定することで、あたかも中心からせり出してくるような表現ができます。

```
[SetBGTrans dst_posX='-400' dst_posY='-300' dst_scaleX='2' dst_scaleY='
2' time='500']
[StartTrans]
```

また、『scaleX』と『scaleY』にマイナスの値を指定すると反転します。たとえば主人公が転倒したシーンで、背景がひっくり返って見えるという演出に使用できるでしょう。

```
[BGLayer surface='PRIMARY' baseX='800' baseY='600' scaleX='-1' scale
Y='-1']

背景画像を上下と左右に反転しました。[p]
```

初登場のパラメータ『baseX』と『baseY』は、サーフェス全体の基準点の座標です。基準点とは少し複雑な概念ですが、「サーフェスのどの座標をそのレイヤーの左上に配置するか」ということです。デフォルトでは「0,0」となっており、背景画像を普通に表示すると、背景画像の左上とウィンドウの左上が一致します。ここで基準点を「400,300」とでもすれば、背景画像のその座標がウィンドウの左上に合わさるようになるのです。

さて背景を反転する場合、基準点を「800,600」（ウィンドウサイズが 800×600 の場合）にします。つまり背景画像の一番右下の部分がウィンドウの左上と合わさります（背景画像がウィンドウの左上方向にはみ出た形になる）。このままでは見た目には何も表示されないのですが、画像を反転させると、ぴったりウィンドウに収まるように表示されます。

●画像を回転する

ただ上下左右に反転させるだけならすでに述べた方法でよいのですが、時間や角度も指定して回転させるには、また別の方法を用います。

```
[SetCharTrans id='0' surface='PRIMARY' baseX='200' baseY='300' posX='200' posY='300' dst_rotation='270' time='1000']
[StartTrans]

キャラクターを１秒で 270° 回転させました。[p]
```

この例で用いるキャラクター画像は横 400、縦 600 ピクセルと仮定します。『baseX』と『baseY』でサーフェスの基準点を「200,300」としていますが、すなわちキャラクター画像の中心部分が基準点になるということです。同時に『posX』と『posY』で表示座標も「200,300」とします。こうすることでキャラクター画像がウィンドウの外にはみ出しません。そしてこのスクリプトを実行すると、キャラクター画像の中心部分を軸に回転します。

ただ、この画面のように足が切れていると見栄えが悪いので、そうならないように画像を作成する必要があるでしょう。

●画像にエフェクトをかける

次は画像の雰囲気を変えるエフェクトについて見てみましょう。ノスタルジックな回想シーンで背景をセピア色にするとか、ショッキングなシーンで反転するとか、いろいろな演出ができます。

```
[BGLayer surface='PRIMARY' colMat='SEPIA']

背景をセピア色にしました。[p]

[BGLayer surface='PRIMARY' colMat='GRAYSCALE']

背景をグレースケールにしました。[p]

[BGLayer surface='PRIMARY' colMat='REVERSAL']

背景をネガポジ調にしました。[p]

[BGLayer surface='PRIMARY' colMat='NONE']

背景を元に戻しました。[p]
```

『colMat』というパラメータは、本来なら色変換行列という難しい知識をもとにした指定が必要なのですが、この例で挙げたように、いくつかのパターンについては特定の文字列を指定すれば済みます。ほとんどの場合は、この 4 つだけで事足りるでしょう。

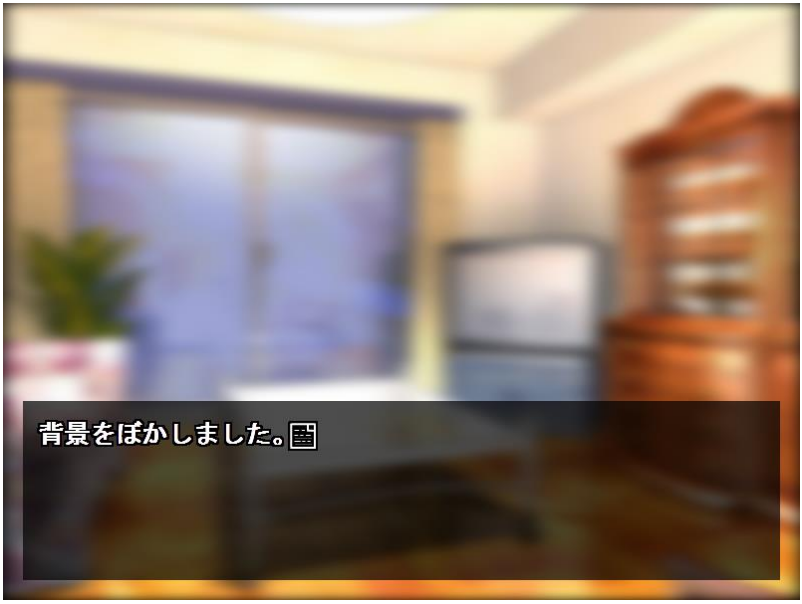
もちろん時間をかけながら変化させることもできます。

```
[SetBGTrans surface='PRIMARY' dst_colMat='REVERSAL' time='1000']
[StartTrans]

背景を 1 秒かけてネガポジ調にしました。[p]
```

他に有用なエフェクトとしては、ブラー（ぼかし）があります。朝目覚めて視界がぼやけているといったシーンで使えます。

```
[BGLayer surface='PRIMARY' blurFilter='blurH=20 blurV=10 quality=3']

背景をぼかしました。[p]
```

『blurFilter』というパラメータでぼかしの強さを指定します。『blurH』は水平方向のぼかし量、『blurV』は縦方向のぼかし量、『quality』はぼかしの実行回数ですが、これは省略しても差し支えないと思います。大事なのは、これらの値をシングルクォーテーションで囲わなければならないことです。



これもやはり時間をかけて変化させることができます。パラメータは『dst_blurFilter』です。

```
[SetBGTrans surface='PRIMARY' dst_blurFilter='blurH=10 blurV=20' time='2000']
[StartTrans]

背景を２秒かけてぼかしました。[p]
```

『イージングによる画像変化』

●変化の速度を調節する

画像を変化させるときに、イージングという指定ができます。これは変化の速度を調節できる機能です。わかりやすく言うと、始めはゆっくり変化させて、終わりのほうでは速く変化させるということが可能なのです。短距離走でだんだん加速してトップスピードになるイメージを思い浮かべればよいでしょうか。

```
[LoadBG path='./resource/BG/bg01.jpg']
[SetFixedBGTrans mode='SCROLL' dir='UP' easing='-100' time='5000']
[StartTrans]

背景が最初は遅く、最後は速く現れました。[p]
```

もっとも単純な方法が、『easing』パラメータに数値を指定することです。省略するとデフォルトの 0 が入りますが、-100 から 100 の間で指定できます。-100 に近ければ「最初は遅く、最後は速く」になり、100 に近ければ「最初は速く、最後は遅く」になります。

●画像の移動などにも使える

前項で取り上げた画像の移動、倍率変化、回転もイージングを使用できます。

```
[SetCharTrans id='0' dst_posX='400' dst_posY='100' eas_posX='-100' eas_
posY='100' time='500']
[StartTrans]

X 座標 400、Y 座標 100 に 0.5 秒で移動しました。[p]
```

キャラクター移動にイージングを使用しました。『eas_posX』と『eas_posY』は座標ではなくイージング値です。試してみればわかるのですが、普通に移動させるよりもかなり面白い、独特の動きになります。『eas_posX』と『eas_posY』を異なる値にするのがコツです。

```
[SetBGTrans dst_scaleX='2' dst_scaleY='2' eas_scaleX='100' eas_scaleY='-100' time='1000']
[StartTrans]

背景画像を１秒で２倍に拡大しました。[p]
```

倍率変化はこの通りです。『eas_scaleX』と『eas_scaleY』に違う値を指定することで、ぐにょんと歪ませながら拡大させることができます。

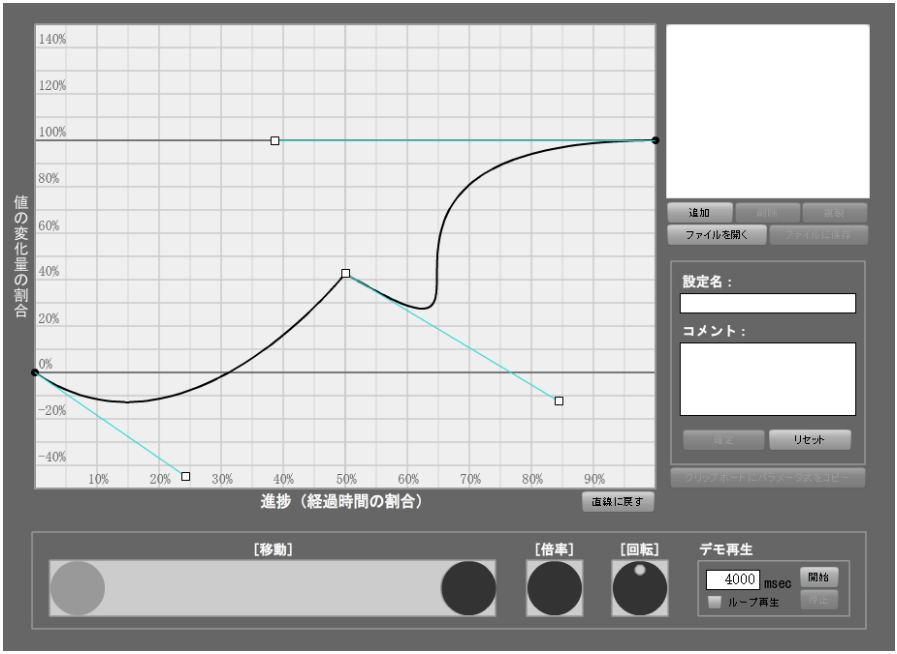
```
[SetCharTrans id='0' surface='PRIMARY' baseX='160' baseY='320' posX='160' posY='320' dst_rotation='180' eas_rotation='-100' time='1000']
[StartTrans]

キャラクターを１秒で 180° 回転させました。[p]
```

回転でイージングを使うときは『eas_rotation』パラメータを用います。何度も値を調節してぴったり合う速度変化を見つけましょう。

●カスタムイージングを使用する

イージングには単に値を指定するだけでなく、より複雑な変化を実現するカスタムイージングという方法があります。

カスタムイージングを使用するには、まず公式サイトの「スクリプト解説」ページにアクセスし、「トランジション（リスト）について」の項目から専用の作成ツールをインストールしてください。



起動すると、直線のグラフが表示されます。これをマウスでドラッグすると、曲線になります。これが変化の割合を視覚的に表現したものです。

曲線上でCTRLキーを押しながらクリックするとアンカーポイントを追加でき、さらに複雑な線にすることもできます。

とりあえずはいろいろ試してみて、デモ再生をしましょう。移動、倍率、回転それぞれの項目で、どのように変化するのかチェックできます。

納得いく曲線データができたら、設定名を入力して「確定」をクリック。あとは保存して定義ファイルにしましょう。保存時に txt と拡張子をつけてください。

このファイルを専用コマンドの[LoadEasingInfo]で読み込みます。

```
[LoadEasingInfo path='./script/def_easing.txt']

イージング定義ファイルを読み込みました。[p]

[SetCharTrans id='0' surface='PRIMARY' dst_posY='500' eas_posY='BOUND' time='1000']
[StartTrans]

ボールが跳ねるようにキャラクターが動きました。[p]
```

『path』パラメータにカスタムイージングの定義ファイルを指定します。そして BOUND という設定名で保存した、ボールがバウンドするような曲線データを『eas_posY』に指定したわけです。

カスタムイージングの設定情報を登録するには、もうひとつ方法があります。

```
[RegistEasingInfo name='BOUND' segList='0,0,0.5,0,0.5,1,0.5,0.5,0.75,0.5,0.75,1,0.75,0.75,0.9,0.75,0.9,1,0.9,0.9,0.975,0.9,0.975,1,0.975,0.975,1,0.975']
```

曲線データを数値に書き直すと、『segList』パラメータに指定しているようになります。先程のツールの「クリップボードにパラメータ式をコピー」から、このパラメータ式を取得することができます。つまり定義ファイルを作成して読み込まなくてもいいのですが、記述が煩雑になるので本書は定義ファイルの読み込みのほうを推奨します。

『文字列を処理する』

●文字列の部分取得をする

LemoNovel AS3 では文字列をいろいろといじることができます。たとえば文字列の一部分だけを表示することができます。

```
[GetStr srcStr='LemoNovel' stPos='4' length='5']
```

文字列の部分取得をする[GetStr]コマンドです。『srcStr』に元の文字列を、『stPos』に取得する文字列の開始位置を（先頭は 0。つまりこの例では 5 文字目から）、『length』に取得したい文字数を指定します。これで「Novel」という文字列が表示されます。

もちろんこれでは「普通に Novel と書けばいいじゃないか」という話になるわけで、変数と組み合わせることで有効に使えるようになります。

```
[Var str='LemoNovel']

[GetStr srcStr="@user.str" stPos='4' length='5']
```

これでも「Novel」と表示されました。

また、部分取得した文字列を変数に格納することもできます。

```
[Var str='LemoNovel']

[GetStr srcStr="@user.str" stPos='4' length='5' var='new_str']
```

ユーザー変数「new_str」に「Novel」が格納されました。『var』パラメータを使用した場合は、部分取得した文字列は表示されません。

これを応用すれば、主人公の名前を自由に決められるタイプのゲームで、名前の一文字目だけを抜き出して「○くん」とヒロインに呼ばせることができます。他にも使いどころはいろいろ考えられるでしょう。

●文字埋めによる文字揃え

たとえば「123」と数を記述するとします。このとき「0123」としたほうが見栄えがいいことがあります。これが文字埋めによる文字揃えで、[PadStr]というコマンドを使用します。

```
[PadStr srcStr='123' align='RIGHT' length='4' padChar='0']
```

『align』は文字列を揃える位置の指定で、『srcStr』で指定した文字列が埋める文字に対してどの位置になるかということです。『length』は文字埋め後の文字数、そして『padChar』が埋める文字。

結果、このスクリプトでは「0123」と表示されます。しかしこれもまた、変数と組み合わせることではじめて威力を発揮します。

```
[Var str='123']

[PadStr srcStr="@user.str" align='RIGHT' length='4' padChar='0']
```

同じように「0123」と表示されましたが、こちらのほうが格段に応用が利きます。シミュレーション系のゲームなどでは、数字の桁数を統一したいことはよくあるものです。

なお省略していますが、このコマンドでも『var』パラメータで文字列を変数に格納できます。

『数値を処理する』

●乱数を取得する

一定の確率で成功か失敗か決まる、どのシーンに移動するか完全に運任せ……そういったシナリオを作りたい場合は、条件分岐に使用する変数をランダムに決める処理が必要です。

```
[Random min='5' max='10' rslt='tmpVal']

取得した数字は[Output msg="@user.tmpVal"]です。[p]
```

乱数、すなわちランダムな数字を取得する[Random]コマンドです。『min』と『max』は発生させる乱数の最小値と最大値で、『rslt』は取得した乱数を格納するユーザー変数名です。この例では 5 から 10 までのいずれかの数字がユーザー変数「tmpVal」に入る、ということになります。

```
[Random min='1' max='100' rslt='tmpVal']

[If exp="@user.tmpVal >= 1 && @user.tmpVal <= 10"]
10%の確率でここに移動します。[p]
[ElseIf exp="@user.tmpVal >= 11 && @user.tmpVal <= 30"]
20%の確率でここに移動します。[p]
[ElseIf exp="@user.tmpVal >= 31 && @user.tmpVal <= 60"]
30%の確率でここに移動します。[p]
[ElseIf exp="@user.tmpVal >= 61 && @user.tmpVal <= 100"]
40%の確率でここに移動します。[p]
[Endif]
```

乱数を使って条件分岐させる一例です。1 から 100 までの乱数を取得し、それに応じて[If]コマンド以下の処理でテキストが変わるようにしています。「○%の確率で○○する」という場合は、こうした記述が基本になります。

●数学関数による数値処理

次は主にシミュレーション系で用途が広い数学関数です。関数というとなんだか難しそうですが、内容は中学生の数学程度です。これには[Math]というコマンドを使います。

```
[Var tmpVal='-50']

[Math srcVal="@user.tmpVal" type='ABS' var='rsltVal']

出力された数字は[Output msg="@user.rsltVal"]です。[p]
```

『srcVal』に処理させたい数値が格納されている変数を指定します。『type』は処理の種類、『var』は処理した数値を新たに格納する変数名です。

処理の種類は以下の通りです。

パラメータ名	詳細
ABS	絶対値を取得
CEIL	切り上げた値（指定値以上のもっとも近い整数）を取得
FLOOR	切り捨てた値（指定値以下のもっとも近い整数）を取得
POW	べき乗（『srcVal』の『procVal』乗）の計算結果を取得
ROUND	四捨五入（もっとも近い整数）を取得
SQRT	平方根を取得

■ABS

絶対値とは 0 からどれだけ離れているかを示す概念です。例で挙げた-50 は、0 から 50 の距離なので、50 と出力されるわけです。

■CEIL

小数点の切り上げに用いる処理です。10.5 と指定していたら、11 と出力されます。

■FLOOR

小数点の切り下げに用いる処理です。10.5 と指定していたら、10 と出力されます。

■POW

べき乗とは 2 乗、3 乗などのことです。この種類のみ、『procVal』というパラメータを用います。次の例では 100 と出力されます。

```
[Var tmpVal='10']

[Math srcVal="@user.tmpVal" type='POW' procVal='2' var='rsltVal']

出力された数字は[Output msg="@user.rsltVal"]です。[p]
```

■ROUND

小数点以下の数字を四捨五入します。10.3 と指定していたら 10、10.7 と指定していたら 11 と出力されます。

■SQRT

平方根とは 2 乗するとその値になる、元の値のことです。たとえば 9 と指定していたら 3 が出力されます。

これらは普通のノベル・アドベンチャーゲームを作る上ではあまり使用しないでしょうが、覚えておいて損はないものです。

『タグ記述をシンプルにするマクロ』

●マクロはシナリオのスリム化に欠かせない

ここまで様々なスクリプトを紹介してきましたが、もうちょっと簡単にならないのかと思った方が多いでしょう。そこで有効なのがマクロ機能です。「一連のキー操作やプログラムを登録しておいて、好きなときに呼び出せる」というパソコン用語ですが、頻繁に使うコマンドの組み合わせをまとめて、使いやすくするのが LemoNovel AS3 のマクロ機能です。

まずは初期設定（ini）ファイルの「マクロ定義ファイルのパス」をチェックしてください。「def_macro.txt」となっているはずですから、そのファイル名で新規テキストファイルを作成してください（UTF8 にすることを忘れずに）。パス部分が「//」でコメントアウトされているなら外しましょう。新しく作成したそのファイルに、次のように記述します。

```
[Macro name='Cross_ChgBG']
[LoadBG path=%path]
[SetFixedBGTrans mode='CROSSFADE' time=%time skip=%skip|true]
[StartTrans]
[EndMacro]
```

これがマクロの基本的な形です。[Macro]コマンドの『name』パラメータにマクロ名を指定します。

次に登録したい内容を記述します。マクロ内に記述されたコマンドのパラメータは、固定するか固定しないかを選ぶことができます。たとえば[SetFixedBGTrans]コマンドの『mode』パラメータは『CROSSFADE』で固定されていますが、その他は『%path』などとなっています。これはマクロを使用する際に、値を状況に応じて決められることを示しているのです。

```
[Cross_ChgBG  path='./resource/BG/bg01.jpg'  time='1500']
```

実際にマクロを使用するときは、以上のようになります。背景画像のクロスフェードは最低でも 3 行を費やしていましたが、かなりシンプルになったことがわかるでしょう。

あれ、『skip』パラメータは？　と思われたでしょうが、『%skip|true』と指定されていたのは、デフォルトの値、つまり省略したときの値が『true』ということです。この場合、『skip』パラメータは省略しても構わないのですが……。

```
[Cross_ChgBG  path='./resource/BG/bg01.jpg'  time='1500'  skip='false']
```

このように『false』と指定することも可能なのです。

パラメータをすべて固定してしまうと、使用する際の記述は簡単になりますが、それだけ多くのマクロを作成しなければならなくなります。便利なマクロですが、このへんのバランスはしっかり考えなければいけません。

●サンプルのマクロを流用する

第 2 章で試したサンプルには、すでに数多くのマクロが登録されています。そのまま流用して使えるものが多いので、コピーしてしまいましょう。

```
// メニュー機能「メッセージを隠す」
[Macro  name='HideMessage']
[HideMsg]
[EndMacro]

// メニュー機能「履歴をみる」
[Macro  name='DisplayHistory']
[DispHistory]
[EndMacro]
```

これは「メッセージ一時消去」と「履歴閲覧」のマクロです。パラメータのない１行だけのコマンドをマクロにする必要はあるの？　と疑問が湧いてくるかと思います。マクロに登録した処理は、通常のシナリオ内で使用する他にも、ボタンや次章で解説するコンテキストメニュー、システムメニューからの呼び出しでも活用できるのです。

では、ボタンからの呼び出しの例を見てみましょう。

```
// テンプレート作成
[RegistBtnTemp name='system' path_Pic='./resource/Button/systembtn.jpg' font_Color='0xFFFFFF' font_Color_Point='0xFFFF00' font_Color_Press='0xFFFF00']

// システムボタン
[Button id='GRAPHIC' type='FIXED' dstLayer='MSG' caption='Graphic' target='HideMessage' posX='500' posY='300' template='system']
[Button id='HISTORY' type='FIXED' dstLayer='MSG' caption='History' target='DisplayHistory' posX='650' posY='300' template='system']

システムボタンが表示されています。[p]
```

テンプレートを作成し、[Button]コマンドで常駐型ボタンを作ります。『target』パラメータというのが初登場ですが、これに指定したマクロを呼び出せるのです。「Graphic」ボタンを押せばメッセージが消え、一時的に画像のみの画面となります。「History」ボタンを押せば履歴画面が表示されます。ボタンの埋め込み先をメッセージレイヤーにしているため、メッセージレイヤーを消去するこれらの処理では、ボタンも一緒に消えます。

さて、常駐型ボタンは一度押すと無効化されてしまいます。したがって「ボタンを押した瞬間に再び有効にする」という処理が必要になります。マクロにこう書き加えましょう。

```
[Macro name='HideMessage']
[ChgButton id='GRAPHIC' enabled='true']
[HideMsg]
[EndMacro]

[Macro name='DisplayHistory']
[ChgButton id='HISTORY' enabled='true']
[DispHistory]
[EndMacro]
```

[ChgButton]コマンドで各 ID のボタンを有効にするのです。これで繰り返しメッセージ一時消去と履歴閲覧ができます。

サンプルにはこの他、メッセージ速度や音量変更といったマクロが用意されています。これでほとんど完成しているマクロなので、実際のゲームにも問題なく使えます。記述内容はいまいち把握できなくてもいいので、ひとまずは自分の def_macro.txt にコピーしておいてください。

第 4 章

スクリプトを構築しよう・システム編

『セーブとロード』

●まずはセーブポイントの設置

いつでもセーブとロードが可能、それがノベル・アドベンチャーゲームの最低限のシステムです。LemoNovel AS3 でのセーブシステムの構築は、すでに解説したラベルと、シーン名を記述するところから始めます。

```
*page1|オープニング
オープニング１ページです。[p]
*page2|
オープニング２ページです。[p]
*page3|
オープニング３ページです。[p]
*page4|
オープニング４ページです。[p]
*page5|物語の始まり
いよいよ物語が始まります。[p]
```

ラベルの後ろに「|」を加え、続けてシーン名を記述しています。これがセーブ時の名前になります。ページ単位で記述する必要はなく、シーンの変化のタイミングで好きなように記述すれば大丈夫です。

ところで、実際にはページ単位でラベルを配置する必要はありません。シーン名を記述していないラベルを配置している区間がありますが、見栄え上、私はこういう書き方を好んでいるというだけの話です。

セーブポイント（ロードしたときにどこから再開されるか）は[p]の通過時やリンク・ボタンの選択終了時に更新されるようになっています。

●[SaveGame]と[LoadGame]

セーブするときは[SaveGame]、ロードするときは[LoadGame]のコマンドを使用します。先程の例に加筆してみましょう。

```
*page1|オープニング
オープニング１ページです。[p]
[LoadGame  no='0']
*page2|
オープニング２ページです。[p]
*page3|
オープニング３ページです。[p]
*page4|
[SaveGame  no='0']
オープニング４ページです。[p]
*page5|物語の始まり
いよいよ物語が始まります。[p]
```

[SaveGame]と[LoadGame]はともに『no』というパラメータを持ちます。これがセーブ番号で、0 から始まって好きなだけ増やせますが、現実的には 100 ほどで充分でしょう。

さて、何もデータがない初回は普通にプレイできますが、次回以降は「*page1」を通過するとセーブデータのロードが行われ、「*page4」から再開されるのです。

セーブとロードの基本機能は以上になりますが、これは到底、実用的なセーブシステムではありません。セーブとロードは専用画面を呼び出して処理するのが一般的です。その専用画面はどう呼び出すのかといえば、サブルーチンを使用します。今回はボタン機能でサブルーチンを呼び出す方法を解説していきます。

●セーブ・ロード画面の構築

手始めにセーブ・ロード画面を呼び出す常駐型ボタンを表示します。

```
[Button id='SAVE' type='FIXED' dstLayer='MSG' caption='SAVE' target='SaveMode' posX='500' posY='300' template='system']
[Button id='LOAD' type='FIXED' dstLayer='MSG' caption='LOAD' target='LoadMode' posX='650' posY='300' template='system']
```

マクロを呼び出す『target』パラメータに、それぞれ『SaveMode』『LoadMode』と指定しました。

次にマクロ定義ファイルに以下のスクリプトを記述します。

```
// セーブ画面
[Macro name='SaveMode']
[Gosub path='./script/savemode.txt']
[EndMacro]

// ロード画面
[Macro name='LoadMode']
[Gosub path='./script/loadmode.txt']
[EndMacro]
```

セーブ・ロードボタンをクリックすると、サブルーチン機能により savemode.txt と loadmode.txt が呼び出されるようになりました。この中に処理を書き込んでいきます。

ところでセーブ・ロード画面で大切なのは、ゲームの進行に影響がないようにすることです。そのため、不必要な機能は一時的に無効化しておいたり、間違えてメッセージレイヤーを消してしまったりしないようにします。

```
// オートモードをオフにする
[AutoMode OFF]
// 履歴処理を一時的にＯＦＦ
[History OFF]
// メッセージレイヤーを１枚を追加生成する
[AddObj type='MSG' id='SAVE_MENU']
// 現在アクティブなメッセージレイヤーの ID を保存しておく
[Var befActiveId=#MsgLayer.activeId]
// 現在のセーブ設定を保存する
[Var befSave=#GameSts.saveSetting]
// セーブポイントの更新を無効にする
[Save UPD_OFF]
// 生成したメッセージレイヤーをアクティブにする
[ActiveMsgLayer id='SAVE_MENU']
// 生成したメッセージレイヤーを画面サイズにする
[MsgLayer id='SAVE_MENU' visible='true' color_BG='0x000000' font_Siz
e='30' posX='0' posY='0' width=#System.width height=#System.height sp
eed_Normal='0' alpha='1']
[MsgLayer id='SAVE_MENU' surface='BASE' topMost='true']
```

これは savemode.txt の、言わば前処理部分です。[AutoMode]はオートモードに関するコマンドで、これにはパラメータが存在せず、『OFF』にすることでオートモードを停止します。[History]も同様で、『OFF』にするとサブルーチン内のテキストが履歴画面に登録されなくなります。

そうしたらメッセージレイヤーの追加生成です。ID をわかりやすく『SAVE_MENU』としていますが、直後に「現在アクティブなメッセージレイヤーの ID を保存しておく」とあります。これはセーブ画面から抜け出る時に、ちゃんと元のメッセージレイヤーを操作できるようにするための処理です。これを記述しないと、メッセージレイヤーを複数使い分けるようなゲームでは思わぬ不具合が出るおそれがあるのです。

次にセーブ設定に関してです。このサブルーチン内では「セーブポイントの更新を無効にする」必要があるので、その前に「現在のセーブ設定を保存」して、元の画面に復帰した時に不具合が出ないようにします。

そして追加生成したメッセージレイヤーをアクティブにして、画面サイズ等の設定をします。『topMost』というパラメータが出てきましたが、これは『true』を指定すると、すべてのキャラクターレイヤーよりも前に表示されるという機能です。メッセージレイヤーに顔を表示するタイプの作品だと、これを指定しておかないと、セーブ画面が出ても顔が表示されたままということになります。このパラメータは『surface』パラメータが『BASE』（メッセージレイヤー自体を示す）でないと使用できないので、[MsgLayer]コマンドをわざわざ 2 回に分けて記述しています。

```
// このメッセージレイヤーを破棄する
[DelObj type='MSG' id='SAVE_MENU']
// セーブポイントの更新を有効に戻す
[Save @user.befSave]
// 通常のメッセージレイヤーに戻す
[ActiveMsgLayer id="@user.befActiveId"]
// 使用済みの変数を破棄する
[DelVar name='befActiveId,befSave']
// 押下したセーブボタンを有効状態に戻す
[ChgButton id='SAVE' enabled='true']
// 履歴処理を再開
[History ON]

[Return]
```

これが後処理部分になります。[DelObj]コマンドで生成した画面をまるごと削除して、そして前処理部分で保存していた変数を利用することで、セーブ設定を通常状態に戻し、元のメッセージレイヤーを操作できるようにします。

不要になった変数は処理速度の関係上、[DelVar]コマンドで破棄するのが望ましいとされています。現段階では『befActiveId』と『befSave』を指定していますが、実際のセーブ・ロード画面はもっと多くの変数を使用するため、それに合わせて破棄する変数も追加指定することになります。

最後に[ChgButton]コマンドでセーブボタンを有効状態に戻し、履歴処理を再開し、[Return]コマンドで元の画面に復帰します。この前後処理の間に、セーブシステムの本体を記述していくのです。以降はセーブ画面についてのみ説明しますが、ロード画面も[SaveGame]コマンドが[LoadGame]になるくらいで、ほとんど変わりはありません。

```
セーブ画面です。[r]
[r]
[Link id='SAVE1']セーブ 1 [/Link][r]
[Link id='SAVE2']セーブ 2 [/Link][r]
[Link id='SAVE3']セーブ 3 [/Link][r]
[r]
[Link id='BACK']戻る[/Link]
[StartSelect var='selId']

[SaveGame no='1' __cond="@user.selId == SAVE1"]
[SaveGame no='2' __cond="@user.selId == SAVE2"]
[SaveGame no='3' __cond="@user.selId == SAVE3"]
```

リンクにはボタンと同じく、条件分岐させるために ID の値を変数に格納する機能があります。「セーブ１」を選んだら[SaveGame]コマンドで１番目の番号にセーブされ、後処理が始まって元の画面に戻る……という流れです。「戻る」を選んだらセーブせずに戻ることになります。

ただセーブするだけならば記述はこれだけで済むのですが、いつセーブしてもリンクテキストが「セーブ１」などで固定されてしまうので、どうにも不便になってしまいます。ここからもう一工夫が必要です。

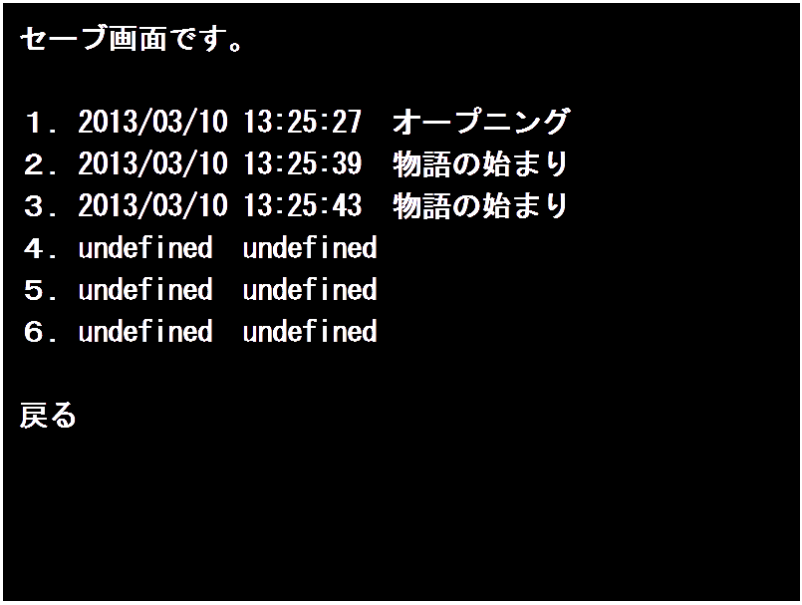
●セーブした時刻とシーン名の表示

セーブした時刻とシーン名を、[Link]～[/Link]の部分に挿入してみましょう。そのためには、セーブ情報の取得のための[GetSaveInfo]コマンドを用います。

```
[GetSaveInfo no='1' date='tmpDate' name='tmpName']
[Var date_name1="@user.tmpDate + '　' + @user.tmpName"]
[Link id='SAVE1']１．[Output msg="@user.date_name1"][/Link]
```

必要なパラメータは 3 つで、『no』はこれまで同様セーブ番号を指定します。『date』と『name』は、セーブ日時とシーン名を格納する変数名をそれぞれ指定します。

次にそれら変数を組み合わせて、新しい変数「date_name1」を作成しています。「'　'」という部分は、セーブ日時とシーン名の間にスペースを挟むためのものです。この変数を[Output]コマンドで文字列として出力します。



この記述を必要なセーブ数に応じて用意し、実際に表示した画面がこれです。セーブデータが存在しない状態では、セーブ日時、シーン名ともに「undefined」となります。これでは見栄えが悪いので、さらに工夫を加えます。

```
[GetSaveInfo no='1' date='tmpDate' name='tmpName' rslt='tmpRslt']
[Var date_name1="@user.tmpDate + '  ' + @user.tmpName" __cond="@user.tmpRslt == true"]
[Var date_name1='----/--/--/--:--:-- セーブデータがありません' __cond="@user.tmpRslt == false"]
[Link id='SAVE1'] 1 . [Output msg="@user.date_name1"][/Link]
```

[GetSaveInfo]に『rslt』というパラメータを記述しています。これにはセーブ情報の取得結果を格納するユーザー変数名を指定します。取得できたら『true』が、取得できなかったら『false』が自動的に入ります。

つまりその番号のセーブデータが存在しない状態（『tmpRslt』が『false』）では、『date_name1』に「----/--/--/--:--:-- セーブデータがありません」と指定しているほうの[Var]コマンドが処理されるのです。

このように[Link]コマンドの前に、セーブスロットの数だけ一連の記述を加えていってください。ずいぶん見栄えがよくなります。

『サムネイルつきのセーブ画面』

●サムネイルを表示するための初期設定

前項でセーブとロードの基本を取り上げましたが、ここから応用に入ります。よりスタイリッシュに見せるには、やはりテキストではなくボタンがいいでしょう。さらにセーブ時点でのサムネイルも見せられれば完璧です。

その前に、セーブ時のサムネイル機能はデフォルトでは無効になっています。初期設定（ini）ファイルを開いて[System]セクションの「セーブデータのサムネイル保持指定」を見てください。

```
// セーブデータのサムネイル保持指定
// [INVALID:無効 UPDATE:セーブポイント通過時の画面 EXEC:セーブ実行時の画面]
save_Thumbnail　= INVALID
```

『INVALID』となっているのを、公式でも推奨されている『EXEC』にしましょう。セーブ実行時の画面が反映されるようになります。さらに以下の指定を加えます。

```
// セーブデータのサムネイルを圧縮する
thumbnail_Compress = false

// サムネイルの縦幅
thumbnail_Height = 75
```

『thumbnail_Compress』は『false』にすると処理が軽くなるので、低スペック環境（主にスマートフォン）を対象にする場合におすすめです。

『thumbnail_Height』はデフォルトでは 75 ですが、もっと大きくしたい場合、小さくしたい場合に調整してください。なお、横幅を指定する『thumbnail_Width』もありますが、省略すれば自動的に画面の縦横比から算出されます。

●ボタンでセーブする

まずはボタンでセーブする際の記述方法を覚えましょう。

```
[RegistBtnTemp name='savebutton1' path_Pic='./resource/Button/saveslot
btn.png' font_Color='0xffffff']
[RegistBtnTemp name='savebutton2' path_Pic='./resource/Button/saveslot
btn.png' font_Color='0x888888']

[GetSaveInfo no='1' name='tmpName' date='tmpDate' rslt='tmpRslt']
[Button id='SAVE1' group='savemode' dstLayer='MSG' dstId='SAVE_ME
NU' caption="@user.tmpDate + '  ' + @user.tmpName" posX='50' posY='
100' template='savebutton1' __cond="@user.tmpRslt == true"]
[Button id='SAVE1' group='savemode' dstLayer='MSG' dstId='SAVE_ME
NU' caption='----/--/-- --:--:--  No Data' posX='50' posY='100' template='s
avebutton2' __cond="@user.tmpRslt == false"]
```

『caption』にセーブ日時とシーン名を指定します。そしてボタンのテンプレートを 2 種類用意しています。

違うのはフォントカラーのみですが、セーブデータがある状態だと『caption』の内容が白文字に、ない状態だと灰色になります。こうした工夫でセーブデータの有無をわかりやすくするのです。

●サムネイルを表示する

それでは先程の例に、サムネイルを表示するための記述を加えます。

```
[RegistBtnTemp name='savebutton1' path_Pic='./resource/Button/saveslot
btn.png' font_Color='0xffffff' align='LEFT' margin_Left='135' posX_SubPi
c='15' posY_SubPic='5']
[RegistBtnTemp name='savebutton2' path_Pic='./resource/Button/saveslot
btn.png' font_Color='0x888888' align='LEFT' margin_Left='135' posX_Su
bPic='15' posY_SubPic='5']

[GetSaveInfo no='1' name='tmpName' date='tmpDate' rslt='tmpRslt']
[Button id='SAVE1' group='savemode' dstLayer='MSG' dstId='SAVE_ME
NU' path_SubPic="$thumbnail.save{1}" caption="@user.tmpDate + '  ' +
 @user.tmpName" posX='50' posY='100' template='savebutton1' __cond="
@user.tmpRslt == true"]
[Button id='SAVE1' group='savemode' dstLayer='MSG' dstId='SAVE_ME
NU' path_SubPic='./resource/Other/nodata.png' caption='----/--/--  --:--:--
No Data' posX='50' posY='100' template='savebutton2' __cond="@user.tm
pRslt == false"]
```

サムネイルは[Button]コマンドの『path_SubPic』パラメータで表示します。これは「画像の中に画像を表示する」という機能で、ここでは通常とは違う「特殊パス指定」という方法を用います。「$thumbnail.save{n}」とは n 番目のセーブデータのサムネイルという意味になります。セーブデータがない場合は、自前で用意した nodata.png を表示するという流れです。

テンプレートでもサムネイル表示に適したパラメータを使用しています。『align』はキャプションの文字揃えで、デフォルトだと中央揃えになっているのを、左揃えにしています。加えてサムネイルを表示するスペースのために、『margin_Left』でボタンの左端からの間隔を指定します。『posX_SubPic』と『posY_SubPic』はそれぞれサムネイルのボタン上での座標です。

セーブ画面です。

2013/03/11 14:33:13　オープニング

2013/03/11 14:33:25　オープニング

No Data　----/--/--　--:--:--　No Data

No Data　----/--/--　--:--:--　No Data

『ループ処理を行う』

●繰り返しスクリプトを実行する

LemoNovel AS3 では、スクリプトを繰り返し実行させる機能があります。これをループ処理といいます。

まずはループ処理の基本から見てみましょう。

```
[Var tmpCnt='1']
以降の処理を、10 回繰り返します。[p]
[While exp="@user.tmpCnt <= 10"]
今、[Output msg="@user.tmpCnt"]回目です。[p]
[Var tmpCnt="@user.tmpCnt + 1"]
[Loop]
```

ループ処理を行う[While]と[Loop]は、必ずセットで使うコマンドです。

最初にユーザー変数「tmpCnt」に 1 を代入します。そして[While]コマンドの『exp』パラメータに条件式を記述しています。この条件式が真である、この場合「tmpCnt」が 10 以内であるなら、[While]～[Loop]内の処理を繰り返すということです。

最終的に条件式が真ではない状態にしないと、延々とループして抜けられなくなりますので注意してください。この例では繰り返す度に「tmpCnt」の値が 1 ずつ増えて、11 に達したら次に進むようにしています。

ただし､[Break]というコマンドを使用することで強制的にループから抜けることもできます。　次の例を､[Break]の頭のコメントがある状態とない状態で試してみてください。

```
[Var tmpCnt='1']
以降の処理を、10 回繰り返します。[p]
[While exp="@user.tmpCnt <= 10"]
今、[Output msg="@user.tmpCnt"]回目です。[p]
[Var tmpCnt="@user.tmpCnt + 1"]
//[Break]
[Loop]
```

●スクリプトを簡略化する

このループ処理が威力を発揮するのは、似たようなスクリプトを繰り返し記述する場合です。サンプルのマクロファイルを覗いてみると、セーブ・ロードで使用していることがわかります。セーブスロットの表示は保存情報のほかはほとんど似たような処理なので、[While]と[Loop]で簡略化しているわけです。

```
[Var tmpCnt='1']
[While exp="@user.tmpCnt <= 5"]
[GetSaveInfo no="@user.tmpCnt" name='tmpName' date='tmpDate' rslt='
tmpRslt']
[Button id="@user.tmpCnt" group='savemode' dstLayer='MSG' dstId='SA
VE_MENU' path_SubPic="$thumbnail.save{@user.tmpCnt}" caption="@us
er.tmpDate + '  ' + @user.tmpName" posX='50' posY="@user.tmpCnt*10
0" template='savebutton1' __cond="@user.tmpRslt == true"]
[Button id="@user.tmpCnt" group='savemode' dstLayer='MSG' dstId='SA
VE_MENU' path_SubPic='./resource/Other/nodata.png' caption='----/--/-- -
-:--:--  No Data' posX='50' posY="@user.tmpCnt*100" template='savebutt
on2' __cond="@user.tmpRslt == false"]
[Var tmpCnt="@user.tmpCnt + 1"]
[Loop]
```

これがループ処理を用いた簡単なセーブ・ロード画面です。

今までは[GetSaveInfo]と、データがある状態とない状態の[Button]コマンドを必要なだけ記述しなければなりませんでしたが、ループ処理を用いることによって、これだけ簡略化できました。

まずユーザー変数「tmpCnt」に 1 を代入して、これが 5 以内ならループするという内容にします。そして[GetSaveInfo]の『no』パラメータ、[Button]の『id』パラメータとサムネイル番号にそれぞれこの変数を指定しています。さらにボタンの Y 座標は「@user.tmpCnt*100」とします。「tmpCnt」が 1 なら 100、2 なら 200 ですね。

これでデータ情報の番号、サムネイル、ボタン座標のそれぞれを変数によって変えた 5 つのセーブスロットが表示されます。

このループ処理は応用範囲が非常に広いので、自分なりに工夫してシステムを構築してみましょう。ちょっと複雑ですが、サンプルのセーブ・ロードが参考になると思います。

『メッセージボックスを表示する』

●システム側からのアクション

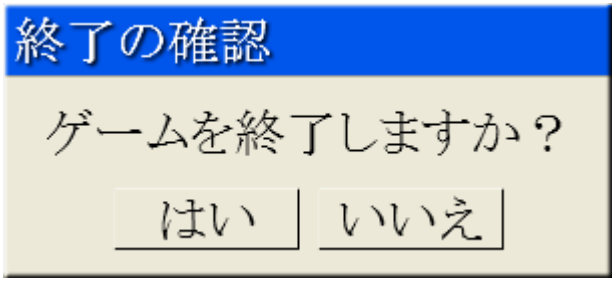
ゲームはユーザーがシステムから問いかけられることがしばしばあります。ゲームを終了するか、タイトル画面に戻るか、セーブするか、どんな名前を入力するか……そんなシーンで使用するのがメッセージボックスです。

メッセージボックスには「はい」「いいえ」を選ばせるタイプと、ユーザーが文字列を入力するタイプのふたつがあります。後者はなくても困りませんが、前者はなければかなり困ることになります。たとえばゲーム終了ボタンをクリックして、本当に終了するかどうかの確認がないと、急に画面が消えてしまうわけです。ユーザーはうっかり操作ミスをすることもありますから、大事な操作をさせる前にワンクッションという意味で、メッセージボックスを表示させることが大事です。

●「はい」「いいえ」タイプ

それでは、ゲーム終了の確認をするメッセージボックスを表示させてみましょう。

```
[MsgBox title='終了の確認' msg='ゲームを終了しますか？' btnCap='はい,いいえ' var='selIdx']

[If exp="@user.selIdx == 0"]
[ExitSystem]
[ElseIf exp="@user.selIdx == 1"]
ゲームを終了しません。[p]
[EndIf]
```



[MsgBox]コマンドの『title』パラメータがタイトルバーに表示するテキスト、『msg』パラメータがユーザーへの問いかけの内容となるテキストです。『btnCap』パラメータはボタンに表示するキャプションで、コンマで区切った分だけボタンの数を増やせます。

そして『var』パラメータには変数名を指定します。各ボタンを押すことで、この変数名に値が格納されます。値は順に 0、1、2……となっており、この例では「はい」ならば 0、「いいえ」ならば 1 が格納されるわけです。次には変数の値によって条件分岐させています。[ExitSystem]がゲームを終了するコマンドになります。

メッセージボックスはフォントの大きさやボタンの間隔など、初期設定（ini）ファイルで細かく調整することができます。メッセージボックスは普通、デザインを状況によって使い分けるということはあまりしないと思いますが、[MsgBox]コマンドのパラメータでメッセージレイヤーと同様に各種の指定をすることができます。

また、ボタンを一切表示せず、システムメッセージとしてのみ使う方法があります。

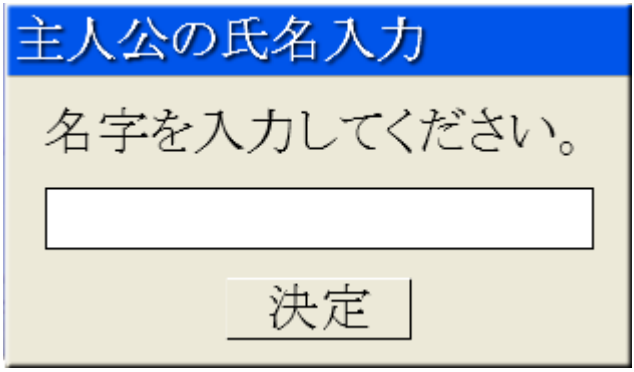
```
[MsgBox title='作者からのメッセージ' msg='このたびは「○○○○」をプレイしていただき、¥r まことにありがとうございました。']
```

メッセージの中に「¥r」というのがありますが、これは改行のための記号です。このようにボタンを省略した場合は、メッセージ部分をクリックすることでメッセージボックスを消去することができます。

●文字列入力タイプ

主人公の名前を自由に決める時などに使用するのが、文字列入力タイプです。インプットボックスともいいます。

```
[MsgBox type='INPUT' title='主人公の氏名入力' msg='名字を入力してください。' btnCap='決定' var_Str='name1']
[MsgBox type='INPUT' title='主人公の氏名入力' msg='名前を入力してください。' btnCap='決定' var_Str='name2']

主人公の氏名は[Output msg="@user.name1 + @user.name2"]です。[p]
```



『type』パラメータを『INPUT』に指定することで、インプットボックスになります。『var_Str』は変数名を指定するパラメータです。この変数に入力した文字列が格納されます。ミステリー系の作品ならば犯人の名前を入力して、その名前によってシナリオを変化させる、という使い方がスタンダードです。

以上が基本的なインプットボックスの使い方ですが、デフォルトの文字列を設定したり、入力する文字列に制限をかけたり、パスワード表示のように演出させることができます。

[MsgBox type='INPUT' title='主人公の氏名入力' msg='名字を入力してください。(5 文字以内)' defStr='山田' maxChars='5' btnCap='決定' var_Str='name1']

デフォルトの文字列を指定するのが『defStr』パラメータで、「山田」という文字列が入力された状態でインプットボックスが表示されます。『maxChars』は最大文字数を指定するパラメータで、5 とすれば 6 文字以上は入力することができません。

[MsgBox type='INPUT' title='キーワード入力' msg='キーワードを入力してください。' restrict='A-Z' btnCap='決定' var_Str='name1']

『restrict』というパラメータは、入力できる文字列を制限します。この例では A から Z までの大文字英字しか入力できません。また、このパラメータの値はシングルクォーテーションで囲む必要があります。文字種の指定方法は以下の表を参考にしてください。

指定する値	概要
abcdefg	「abcdefg」の 7 文字のみ入力可能
a-z	小文字英字のみ入力可能
A-Z	大文字英字のみ入力可能
^0-9	半角数字以外が入力可能
^0-9 A-Z	半角数字と大文字英字以外が入力可能

[MsgBox type='INPUT' title='パスワード入力' msg='パスワードを入力してください。' btnCap='決定' password='true' var_Str='password']

『password』パラメータを『true』にすると、入力した文字列が「******」と表示されます。パソコンでパスワード入力……というシーンで使える演出です。

●メッセージボックスを使っているマクロ

サンプルにはメッセージボックスを使ったマクロがいくつか見られますが、その中から比較的わかりやすいのを取り上げてみます。

```
// メニュー機能「最初に戻る」
[Macro name='ReturnTitle' export=true]
	[If exp='%confirm == true']
		// 確認が必要
		[MsgBox title='「最初に戻る」の確認' msg='現在の進行を中止して最初に戻ります。¥r（セーブされていない情報は破棄されます）¥r¥r よろしいですか？' btnCap=はい,いいえ defBtn=1 var=selIdx]

		[If exp="@user.selIdx == 1"]
			// 「いいえ」を選択
			[ExitMacro]
		[EndIf]
	[EndIf]

	[InitGame path='./script/first.txt' label='Start']
[EndMacro]

[Button id='TITLE' type='FIXED' dstLayer='MSG' caption='Title' target='ReturnTitle' arguments='confirm=true' posX='350' posY='300' template='system']
```

はじめに[If]で条件分岐します。『confirm』が『true』ならば、最初に戻るかどうかの確認をするのですが、『false』ならば確認なしで最初に戻るということです。

[Button]コマンドに『arguments』というパラメータがあります。これにはマクロに渡すパラメータ式を指定します。例のように『confirm=true』と指定し

なければ、最初に戻るかどうかの確認はされません。true にするか false にするかは、制作者が自由に決めていいわけです。

次に[MsgBox]コマンドで確認します。このへんは先述のとおりですが、『defBtn』というパラメータは、メッセージボックスの表示時にフォーカスさせておくボタンを指定します。この例では「いいえ」にフォーカスさせていて、エンターキーを押すだけでマクロを終了する[ExitMacro]が実行され、それより下のコマンドの処理はされずにゲームに戻ります。

確認のメッセージボックスが出なかった場合、または確認で「はい」を選んだ場合、ゲームを初期化する[InitGame]コマンドが実行されます。『path』と『label』の両パラメータに初期化後のジャンプ先を指定しています。

●セーブ時に上書きの確認

データのない番号に最初にセーブする時はともかく、2 度目以降はその前に確認をさせましょう。操作ミスで問答無用に上書きされるようなゲームは、明らかに不親切になってしまいます。そこで上書きの前に……。

```
[StartSelect var='selId']

[If exp="@user.selId == SAVE1"]
	[If exp="@user.tmpRslt == true"]
		[MsgBox title='上書き確認' msg='記録を上書きしますか？' btnCap='はい,いいえ' var='selIdx']
		[If exp="@user.selIdx == 0"]
			[SaveGame no='1']
		[EndIf]
	[Else]
		[SaveGame no='1']
	[EndIf]
[EndIf]
```

メッセージボックスを表示してユーザーに確認させてみます。

・最初にどのセーブ番号を選んだかで条件分岐
・そのセーブ番号にデータがない場合、すぐにセーブする
・すでにデータがあった場合は、メッセージボックスを表示する
・「はい」を選んだらセーブし、「いいえ」を選んだらセーブせずに終了

流れはこのようになります。ロードの際にも、間違って違うデータをロードしてしまわないよう、こうした確認のメッセージを挿入することが好ましいでしょう。

『トーストを表示する』

●簡素なメッセージ表示に特化した機能

トーストとは情報を通知するミニウィンドウで、スマートフォンではよく利用されています。LemoNovel AS3 でも、AIR Mobile 版に限らずこの機能を使うことができます。

```
[Toast msg='トーストを表示しました。']
```

以上が基本的な記述方法です。メッセージボックスと違い、一定時間が経つと自動的にフェードアウトしていきます。

初期設定（ini）ファイルでフォントサイズや背景色を調整できますが、パラメータで指定することもできます。

```
[Toast msg='トーストを表示しました。' color_BG='0x0000FF' time='5000']
```

●表示途中で削除する

トーストは便利な機能ですが、このシーンで表示されていると邪魔、という場合はフェードアウトを待たずに削除することができます。

```
[Toast msg='トーストを表示しました。']

トーストを表示しました。[p]

[Toast priority='0']

トーストを削除しました。[p]
```

『priority』はトーストの優先度を指定するパラメータです。多少扱いが難しいパラメータなので本書では詳しく解説しませんが、0 と指定するのが特殊な方法で、すべてのトーストを削除することができます。

『履歴を制御する』

●メッセージを履歴に表示しない

「セーブとロード」の項ですでに触れていますが、一時的にメッセージを履歴画面に表示したくない場合は[History OFF]とします。

```
[History OFF]
このメッセージは履歴に表示されません。[p]

[History ON]
ここから履歴に表示されます。[p]
```

[History ON]で再び表示されるようになります。タイトル画面などでは必ずOFFにしておきましょう。

●任意の文字列を履歴画面に出力する

逆に、メッセージレイヤーには表示しないが履歴画面には表示するという機能があります。

```
[OutHistory msg='これは履歴のみに出力されるメッセージです。']

履歴画面を表示してみましょう。[p]
```

[OutHistory]コマンドの『msg』パラメータに、自由にメッセージを記述します。あまり多用する機能ではありませんが、シーン名を挿入したり、ヒントを隠しておくといった使い方ができます。

●履歴メッセージを削除する

何らかの理由で履歴メッセージを削除したい場合は、[ClearHistory]コマンドを用います。

```
[ClearHistory]

履歴メッセージを削除しました。[p]
```

このコマンドにはパラメータがなく、そのまま記述するだけです。

●AIR Mobile 版で設定しておくべきこと

履歴画面は初期設定（ini）ファイルでかなり自由なカスタマイズができますが、AIR Mobile 版の作品を作る場合に、必ず調整しなければならない項目があります。それは「戻るボタン」の大きさです。

パソコンであればマウスで簡単にクリックとドラッグができますが、スマートフォンの画面は小さいので、初期設定よりもだいぶ大きくしなければまともに操作できません。そこで[HistoryLayer]の項目に、以下の記述を加えてください。

```
// 戻るボタンの大きさ
defBtn_Width = 50
```

『defBtn_Width』が戻るボタンの縦横サイズです。AIR Mobile 版では最低でも 50 ピクセルはないと厳しいでしょう。スクロールバーの横幅は、ここで設定したサイズと同じになります。

『コンテキストメニューを表示する』

●シンプルなメニューを作りたいなら

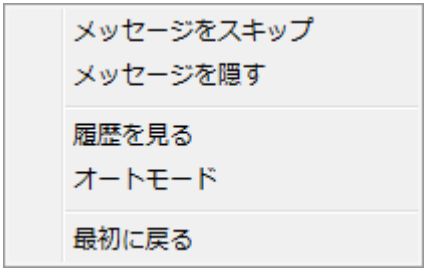
Windows では画面を右クリック、Mac では control＋クリックすると何らかのメニューが表示されます。これをコンテキストメニューといって、同じことが LemoNovel AS3 でも可能です。

デザイン性には乏しいですが、ボタンを配置せずともメッセージ一時消去や履歴閲覧ができるようになるので、お手軽さを重視する人向けです。なお、このメニューは AIR Mobile 版では表示できません。

```
[SetContextMenu id='1' caption='メッセージをスキップ' target='MsgSkip' visible='true']
[SetContextMenu id='2' caption='メッセージを隠す' target='HideMessage' visible='true']
[SetContextMenu id='3' caption='履歴を見る' target='DisplayHistory' visible='true' separator='true']
[SetContextMenu id='4' caption='オートモード' target='ChgAutoMode' visible='true']
[SetContextMenu id='5' caption='最初に戻る' target='ReturnTitle' arguments='confirm=true' visible='true' separator='true']
```

コンテキストメニューを実装するには[SetContextMenu]コマンドを使用します。『id』は 0 から 15 までの範囲で指定できます。0 は特別扱いで右クリックイベントの設定に用いるのですが、これは後ほど解説します。

『caption』と『target』と『arguments』はボタン機能と同様です。デフォルトでは非表示なので『visible』を最初は必ず『true』にします。



『separator』は区切り線の有無を指定するパラメータで、『true』にすると直前にそれが表示されます。単に見栄えの問題なので、必須というわけではありません。以上がコンテキストメニューの基本となります。

●条件によって項目を非表示・無効化

たとえばメッセージを一時消去している状態でも、コンテキストメニューでは「メッセージを隠す」の項目は表示されます。当然、再び選択したところで意味はありません。

そのような無意味な操作をユーザーにさせたくない場合は、「この状態ならこの項目は非表示か無効化する」と設定できます。

```
[SetContextMenu id='1' caption='メッセージをスキップ' target='MsgSkip'
 visible='true' syncVisible='HIDEMSG']
[SetContextMenu id='2' caption='メッセージを隠す' target='HideMessage'
 visible='true' syncEnabled='DOING|HIDEMSG']
```

特定の条件で項目を非表示にするのが『syncVisible』で、無効化するのが『syncEnabled』になります。

この例では「メッセージをスキップ」の項目に『syncVisible=HIDEMSG』と記述し、メッセージ一時消去時に表示させないようにしています。また「メッセージを隠す」の項目は『syncEnabled='DOING|HIDEMSG'』と記述し、スクリプト進行中（テキストが流れている最中など）とメッセージ一時消去時に無効化しています。

同一の項目に対して『syncVisible』と『syncEnabled』の両方を設定することもできます。条件の種類は下の表のとおりで、「|」で区切ることで複数指定が可能です。

『syncVisible』と『syncEnabled』に指定できる条件	
DOING	スクリプト進行中
HIDEMSG	メッセージ隠し中
SELECT	選択モード中
AUTOMODE	オートモード中
HISTORY	履歴画面表示中
INVALID_SAVE	セーブ不可中
NOTREAD	未読区間中
LIMIT_PROGOPE	進行操作制限中
LIMIT_DISPHIST	履歴画面表示制限中
LIMIT_SKIP	スキップ制限中

●右クリックイベント

[SetContextMenu]コマンドの ID を 0 に指定することで、指定したマクロを右クリックで呼び出すことができます。

```
[SetContextMenu id='0' target='HideMessage']
```

こう記述すると､右クリックでメッセージが一時消去されるようになります。使用する頻度の高い処理を登録しておき、システムボタンと併用しましょう。

なお、AIR Mobile 版では右クリックという操作はありませんが、2 本指タップが右クリックとして処理されるようになっています（ただし対応しているハードのみ)。

『システムメニューを表示する』

●画面上部のメニュー

システムメニューとは一般的に、ウェブブラウザやアプリケーションの上部に並ぶメニューを指します。ノベル・アドベンチャーゲームでも「マウスを上部に持っていくとメニューが現われる」というシステムを採用している作品がよく見られます。LemoNovel AS3 でもこうしたシステムメニューの構築ができるのです。

これを実装するには、システムメニュー定義ファイルを作成します（コンテキストメニューのように 1 行ずつコマンドを記述する方法もあるのですが、定義ファイルのほうが簡単に済みます)。サンプルに def_sysmenu.txt というファイルがありますので、これを開いてみましょう。

```
[Settings]
enabled=true
//font_Name=ipa_gothic
//font_Embed=true
//font_Size=8
```

まずこのような記述があります。定義ファイルは通常のコマンド記述とは、いささか違っているのです。[Settings]からの区間には、システムメニューの設定を記述していきます。『enabled』はシステムメニューの有効無効を指定するもので、何よりこれを『true』にしなければ始まりません。

コメントアウトされている部分は、いずれもメニューのフォントを設定するパラメータです。こういったフォント設定やメニューバーの余白、背景色などを自由に決められます。コマンドリファレンスの[SetSysMenu]にパラメータの一覧がありますので、チェックしてください。

●メインメニューとサブメニュー

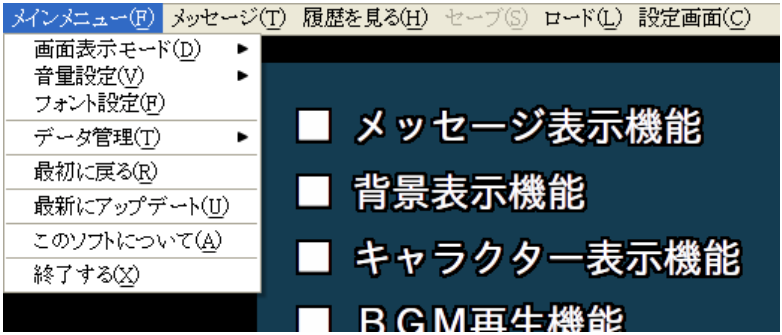
その下の[Items]からの区間が、システムメニューの根幹部分になります。

```
[Items]
// メニューバー
type=SUBMENU caption='メインメニュー' target='MENU_MAIN' scut_Key=F
type=SUBMENU caption='メッセージ' target='MENU_MSG' scut_Key=T
type=NORMAL caption='履歴を見る' target='DisplayHistory' syncEnabled=DOING|LIMIT_DISPHIST scut_Key=H
type=SUBMENU caption='セーブ' target='MENU_SAVE' syncEnabled=INVALID_SAVE scut_Key=S
type=SUBMENU caption='ロード' target='MENU_LOAD' scut_Key=L
type=NORMAL caption='設定画面' target='DispConfig' arguments='menuType=SETTINGS' syncEnabled=DOING scut_Key=C
```

以上の 6 つの項目がメニューバーに登録されています。『menuId』パラメータでメニューID を指定するのですが、『root』と指定するかこの例のように省略すると、メニューバーに項目が登録されるのです。

次に『type』パラメータで追加する項目の種別を指定します。「設定画面」のみ『NORMAL』となっていますが、この項目は単独で起動します。他の『SUBMENU』となっているのは、ツリー状にサブメニューが表示される項目です。

『target』パラメータの扱いは、『type』パラメータの指定が『NORMAL』か『SUBMENU』かで違いがあります。『NORMAL』ならば指定したマクロが実行されますが、『SUBMENU』だとサブメニューのメニューID を指定したことになるのです。



『SUBMENU』を指定している「メインメニュー」をクリックすると、この画像のようにサブメニューが現われます。そのサブメニューの記述を一部抜粋します。

```
menuId=MENU_MAIN type=SUBMENU caption='画面表示モード' target
='MENU_SCREENMODE' separator=true scut_Key=D
menuId=MENU_MAIN type=SUBMENU caption='音量設定' target='MEN
U_VOLUME' scut_Key=V
menuId=MENU_MAIN type=NORMAL caption='フォント設定' target='Sh
owFontDlg' syncEnabled=DOING scut_Key=F
```

再び『type』パラメータが『NORMAL』と『SUBMENU』で分かれています。「フォント設定」をクリックするとフォント設定画面が表示されますが、「画面表示モード」と「音量設定」は、マウスカーソルを合わせるとさらにサブメニューが横に展開されるのです。「音量設定」は BGM や効果音の項目に分かれており、さらに各音量設定のためのサブメニューが用意されている……という具合です。

ユーザビリティを考えると、何回もサブメニューを展開させるように構築するのは好ましくありません。「音量設定」のように、最大でせいぜい 3 回が限度でしょう。

●ショートカットの設定

これまでの例ではいずれも『scut_Key』というパラメータがありました。システムメニューはマウスだけでなく、キーボード操作でも表示することが可能で、その項目が表示されている状態で指定されたアルファベットを押下すると、その項目が実行されます。

指定できるのはAからZまでのアルファベットと、0から9までの数字です。ただしアルファベットは M を除きます。というのも、M キーにはシステムメニューの表示と非表示を切り替える機能が割り当てられているためです。

したがって、最初に M キーを押し、そこから F キーを 2 連続で押していけば、フォント設定画面が表示されます。キーボードのない Android と iPhone ではショートカット機能は使えませんが、Android の場合、メニューキーの押下が M キー押下と同じ処理になっています。

●定義ファイルの読み込みと保存情報の復元

システムメニューの構築が終わったら、定義ファイルをシナリオ内で読み込みます。

```
[LoadSysMenu  path='./script/def_sysmenu.txt']
```

これでいつでもシステムメニューを表示できますが、場合によっては非表示にしたいこともあります。

```
[SetSysMenu  enabled='false']
```

こう記述すれば、システムメニューは表示されなくなります。

『ブラウザに接続する』

●自分のサイトに案内する

ゲームのプレイ中、あらすじやキャラクターを確認するため、ブラウザを起動して制作者のサイトにアクセスすることはよくあります。そこでユーザーの手間を省くためにゲーム内から直接ブラウザを起動させましょう。

```
[GetURL url='https://www.google.co.jp/' target='_blank']
```

[GetURL]コマンドの『url』パラメータに、サイトのアドレスを指定します。『target』パラメータはページをどのように開かせるかという指定です。

パラメータ	対象となるウィンドウ
_self	現在のウィンドウ内の現在のフレーム
_blank	新規ウィンドウ
_parent	現在のフレームの親フレーム
_top	現在のウィンドウ内の最上位のフレーム

省略すると『_blank』になります。特に理由がない限りはこの指定が一番わかりやすいでしょう。したがってこのパラメータは省略してもかまいません。

『各種鑑賞モードを作る』

●CG 鑑賞モードの作り方

ノベル・アドベンチャーゲームのクリア後の楽しみといえば CG、音楽、エンディング等の鑑賞モードです。これらの機能はゲームが終了しても保持されるシステム変数を用いて実装します。

まず CG 鑑賞モードです。イベント CG を表示する際に[SysVar]コマンドを記述してください。

```
[LoadBG surface='PRIMARY' path='./resource/Event/cg01.jpg']
[SysVar cg01='true']
[SaveSystem]
```

ここで[SaveSystem]というコマンドが初登場しています。実はシステム変数は[SysVar]コマンドだけでは不十分で、[SaveSystem]コマンドを用いることでゲームシステムに保存されます。これを記述しないと、ゲーム終了時に設定の変更情報が失われてしまうのです。

そしてゲームの冒頭部分で、[SaveSystem]と対になる[LoadSystem]コマンドが必要です。

```
[LoadSystem]
```

これによって、保存されたシステム情報が復元されます。各種鑑賞モードを作るなら、この保存と復元のコマンドを必ずセットで使用してください。

さて、システム変数「cg01」を「true」としました。これでボタン表示の際に条件分岐させます。

```
[Button id='CG1' group='cgmode' path_Pic='./resource/Button/btn_cg1.png
' posX='140' posY='60' __cond="@sys.cg01 == true"]
[Button id='CG1_No' group='cgmode' path_Pic='./resource/Button/btn_noi
mage.png' posX='140' posY='60' enabled='false' __cond="@sys.cg01 != tru
e"]
[StartSelectBtn group='cgmode' var='selId' delBtn='true']

[Goto label='cg1' __cond="@user.selId == CG1"]
```

これが記述の一例です。システム変数「cg01」が「true」ならば btn_cg1.png というボタン画像（イベント CG のサムネイル）を表示し、クリックするとラベル「cg1」にジャンプします。「true」でないならばまだその CG を見ていないことを示す btn_noimage.png を表示するという流れです。

また、クリア後にすべてのイベント CG を見られるようにする、というタイプのゲームもあります。その場合は、CG 鑑賞モードへ移動するためのリンク自体を最初は非表示にして、クリアと同時に開放するようにするとよいでしょう。

●音楽鑑賞モードの作り方

音楽鑑賞モードも、CG とたいして違いはありません。初めてその曲を再生するシーンで[SysVar]コマンドを記述してください。

```
[LoadBGM buffer='PRIMARY' path='./resource/BGM/bgm01.mp3']
[PlayBGM buffer='PRIMARY' mode='PLAY']
[SysVar bgm01='true']
[SaveSystem]
```

システム変数「bgm01」を「true」としました。これでボタン表示の際に条件分岐させます。

```
[Button id='BGM1' group='musicmode' path_Pic='./resource/Button/btn_bgm.png' caption='BGM01' posX='100' posY='60' __cond="@sys.bgm01 == true"]
[Button id='BGM1_NO' group='musicmode' path_Pic='./resource/Button/btn_bgm.png' caption='？？？？' posX='100' posY='60' __cond="@sys.bgm01 != true"]
[StartSelectBtn group='musicmode' var='selId' delBtn='false']

[Goto label='bgm1' __cond="@user.selId == BGM1"]
```

これはボタン画像をすべて統一して、キャプションで曲名を表示するという方法です（まだ聞いていない曲は曲名がわからない）。このほうが手間はかからないでしょう。

●エンディングリストの作り方

何らかのエンディングを迎えた際に、エンディングリストに記録されるようにします。

```
エンディング 1 [p]

[SysVar ending01='true']
[SaveSystem]

[InitGame path='./script/first.txt' label='start']
```

タイトルに戻る処理をする前に、エンディング情報を保存します。以降は CG、音楽と同じです。

```
[Button id='END1' group='endinglist' caption='1:都市伝説' posX='70' posY='50' __cond="@sys.ending01 == true"]
[Button id='END1_No' group='endinglist' caption='1:？？？？' posX='70' posY='50' __cond="@sys.ending01 != true"]
```

これはボタン画像を使わない、キャプションのみのボタンです。また、回想機能を採用せず、エンディング名を表示させたいだけなら、ジャンプ処理を記述する必要もありません。

『データを削除する』

●セーブデータ範囲を指定して削除する

必要のないセーブデータを削除する……ユーザーにとって必須とは言えませんが、かゆいところに手が届くという感じの機能です。次のスクリプトを、リンクやボタンで選んで実行できるようにしてみましょう。

```
[DelSave stNo='1']

セーブデータ番号１を削除しました。[p]
```

[DelSave]コマンドの『stNo』パラメータに、削除対象のセーブデータ番号を指定します。『stNo』は正確には範囲の開始値で、終了値の『edNo』パラメータと合わせて記述することで、特定の範囲のセーブデータをまとめて削除することができます。

```
[DelSave stNo='1' edNo='5']

セーブデータ番号１から５までを削除しました。[p]
```

●システムデータを削除する

[SysVer]コマンドで生成した変数など、システム系のデータを削除したい場合は、[DelSysSave]コマンドを使用します。

```
[DelSysSave]

システムセーブデータを削除しました。[p]
```

[DelSysSave]コマンドには 2 種類のパラメータがあるのですが、省略しても差し支えありません。このコマンドを実行した場合は、不具合を避けるために［ExitSystem］でゲームを終了するか、［InitGame］で起動直後の状態に戻すことが推奨されます。

●すべてのデータをまとめて削除するマクロ

サンプルのマクロを覗いてみると、全データ初期化処理の「InitializeData」が登録されています。これを実行すると個別のセーブデータ、システムデータともに削除されます。自分のマクロファイルにコピーして使ってみましょう。

```
// 全データ初期化処理
[Macro name='InitializeData']
	[MsgBox title='全データ初期化の確認' msg='セーブデータを含む全てのデータを破棄し、最初に戻ります。¥r 処理を続行してよろしいですか？' btnCap=はい,いいえ defBtn=1 var=selIdx]
	[If exp="@user.selIdx == 1"]
		// 「いいえ」を選択
		[ExitMacro]
	[EndIf]

	// ゲームセーブデータを削除する
	[DelSave stNo=0 edNo=100]

	// システムセーブデータを削除する(データもクリアする)
	[DelSysSave init=true]

	// 最初に戻す
	[InitGame path='./script/first.txt']
[EndMacro]
```

この機能が役立つのは、ユーザーよりもむしろ制作者です。全データを削除して最初から見直すというのは、制作中にしょっちゅうあることですので。

『フォントを埋め込む』

●作品の雰囲気を変えるフォント

LemoNovel AS3 では基本的なゴシック体、明朝体などのほか、あらゆるフォントを使用することができます。特に、ユーザーがコンピュータにインストールしていないフォントでも表示されるように、フォントの埋め込み機能があります。

サンプルには「ipag」というフォントデータが同梱されていますので、これを例に見てみましょう。

```
// フォントの読み込みを開始する
[LoadFont path='./ipag.swf' defName='IPAG_Normal']
```

first.txt の最初にこのような記述があります。[LoadFont]コマンドの『path』に、画像データなどと同じようにフォントデータのパスを指定します。『defName』については後述します。

フォントの指定は[MsgLayer]コマンドでも行うことができます。

```
[MsgLayer font_Name='ipa_gothic' font_Embed='true' font_Name_Rb='ipa_gothic' font_Embed_Rb='true']
```

『font_Embed』と『font_Embed_Rb』は、通常メッセージとルビに埋め込みフォントを使うかのパラメータです。ともに埋め込みフォントを使用する際に『true』とすることが必須となります。

次に初期設定（ini）ファイルの[MsgLayer]セクションを見てください。

```
// メッセージのフォント
font_Name = "ipa_gothic"
//font_Name       = "ＭＳ ゴシック,Osaka,IPA ゴシック"

// メッセージの埋め込みフォント使用有無
font_Embed      = true
//font_Embed      = false
```

このようにすることで、埋め込みフォントが表示されるようになります。

実際に見てみると、埋め込みフォントのほうが格段に綺麗に表示されています。

また、デフォルトでは Windows は「ＭＳ ゴシック」、Mac では「Osaka」を使用するようになっているのですが、微妙な表示の違いからレイアウトが崩れる可能性もあります。たとえば Windows ではテキストが 1 ページにしっかり収まっていたのに、Mac では 1 ページに収まりきらない、などです。こうした観点からも、環境に左右されない埋め込みフォントを使用するメリットがあります。

●フォントデータの作成方法

ネット上で手に入れたユニークなフォントを使ってみたい、そう思ったらipag のような埋め込みフォントデータを作成します。ここではフリーフォントの「みかちゃんフォント」を例に作成していきます。

まずはフォントの埋め込みに必要な Flex SDK をインストールします。

【Download Adobe Flex SDK】

http://www.adobe.com/devnet/flex/flex-sdk-download.html

本書では C ドライブの直下に解凍したとして解説を進めます。解凍したら「C:¥flex_sdk¥」内に「mika_font」フォルダを作成して、その中に ttf ファイルを入れてください。

次にこのフォルダ内に、フォントの埋め込みを行うためのテキストファイルを作成し、以下のとおり記述します。

```
package{

    import flash.display.Sprite;

    public class Mika extends Sprite
    {
        [Embed(source="mika.ttf", fontName="mika_font", mimeType="a
pplication/x-font", embedAsCFF="true", advancedAntiAliasing="true")]
        public static const Normal:Class;

        public function Mika()
        {

        }
```

```
    }
}
```

内容はプログラミング言語ですが、細かい箇所は覚えなくてかまいません。抑えておくのは「public class」ではじまる部分からです。ここで「public class Mika extends Sprite」とします。「Mika」の部分はそれぞれのフォントに合わせて自由に決めてください。

そして「Embed」以下、『source』に先程入れたフォントのパスを、『fontName』に LemoNovel AS3 で利用するフォントの識別子を指定します。「public static const」は自由ですが『Normal』とします。「public function」には「public class」と同じ名前を指定します。

あとはこのテキストファイルの名前もクラス名と同じにして、かつ拡張子を「as」にして保存します。「Mika.as」となるわけです。

なお、ひとつのファイルに複数のフォントを埋め込むこともできます。[Embed]以下を必要な分記述します。

```
package{

    import flash.display.Sprite;

    public class Mika extends Sprite
    {
        [Embed(source="mika.ttf", fontName="mika_font", mimeType="a
pplication/x-font", embedAsCFF="true", advancedAntiAliasing="true")]
        public static const Normal:Class;

        [Embed(source="mika-P.ttf", fontName="mika-P_font", mimeType
="application/x-font", embedAsCFF="true", advancedAntiAliasing="true
")]
```

```
        public static const Pro:Class;

        public function Mika()
        {

        }
    }
}
```

次はコンフィグファイルというものを用意します。as ファイルをもとにして埋め込みフォントのデータファイルを生成するためのテキストファイルです。これに以下の内容を記述してください。

```
<?xml version="1.0" encoding="UTF-8" ?>
<flex-config>
        <output>Mika.swf</output>
        <default-frame-rate>30</default-frame-rate>
        <static-link-runtime-shared-libraries>true</static-link-runtime-shared-libraries>
</flex-config>
```

<output>～</output>で括られた箇所に、埋め込みフォントデータのファイル名を、拡張子「swf」で指定します。調整するのはこれだけです。

このテキストファイルを、拡張子を「xml」にした「Mika-config.xml」として保存してください。以上で準備が完了です。

●ファイルを生成する

生成にはコマンドプロンプトを使用します。Windows メニューの「プログラムのファイルと検索」に「cmd」と入力して起動してください。Mac の場合は「アプリケーション」→「ユーティリティ」から「ターミナル」を起動します。

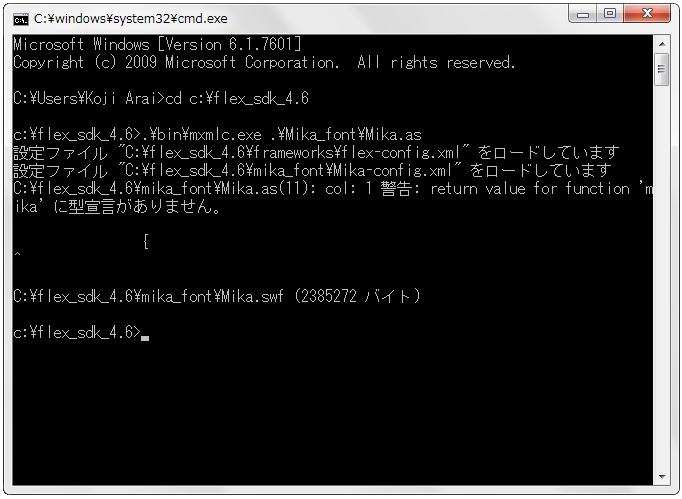
画面が出たら、最初にカレントディレクトリを移動します（操作対象のフォルダに移るという意味です)。コピー&ペーストで次のように入力してください。

```
cd c:¥flex_sdk_4.6
```

※Windows で Flex SDK のフォルダ名が「flex_sdk_4.6」の場合

これで「c:¥flex_sdk_4.6>」となるので、次のように続け、エンターキーを押せば、データファイルが生成されます。

```
c:¥flex_sdk_4.6>.¥bin¥mxmlc.exe .¥Mika_font¥Mika.as
```



最後におさらいも兼ねて、フォントの埋め込みデータファイルを使ったスクリプトの追加説明をします。

```
[LoadFont path='./Mika.swf' defName='Mika_Normal']
[MsgLayer font_Name='mika_font' font_Embed='true' font_Name_Rb='mika_font' font_Embed_Rb='true']
```

[LoadFont]コマンドの『defName』は、as ファイルの「public static const」で『Normal』とした部分にクラス名をくっつけたものです。これをフォントの定義名といいます。

```
[LoadFont path='./Mika.swf' defName='Mika_Normal,Mika_Pro']
```

このようにひとつのデータファイルに複数のフォントを埋め込んでいる場合に、定義名をカンマ「,」で連結して指定します。ただ、複数のフォントを埋め込むというのはあまり推奨されません。

そして[MsgLayer]コマンドの『font_Name』と『font_Name_Rb』に、それぞれ識別子を指定します。途中で別の埋め込みフォントに変更するなら、ここを指定し直すのです。

『ファイルをキャッシュ/先読みする』

●AS3 版で必須の機能

ブラウザでプレイする AS3 版は、データをネット上で読み込みます。必然的に読み込み時間がかかるわけですが、プレイ中にしょっちゅうゲームが停止してしまってはいけません。

そこで使うのが[Cache]コマンドです。必要なファイルをまとめてダウンロードする「キャッシュ」、あるいはスクリプトの進行と同時にダウンロードする「先読み」の 2 種類の機能があります。

```
[Cache path='./BG/sampleBG1.jpg']
[Cache path='./Char/sampleChar1.swf']
[Cache path='./BGM/sampleBGM1.mp3']

[StartCache]
```

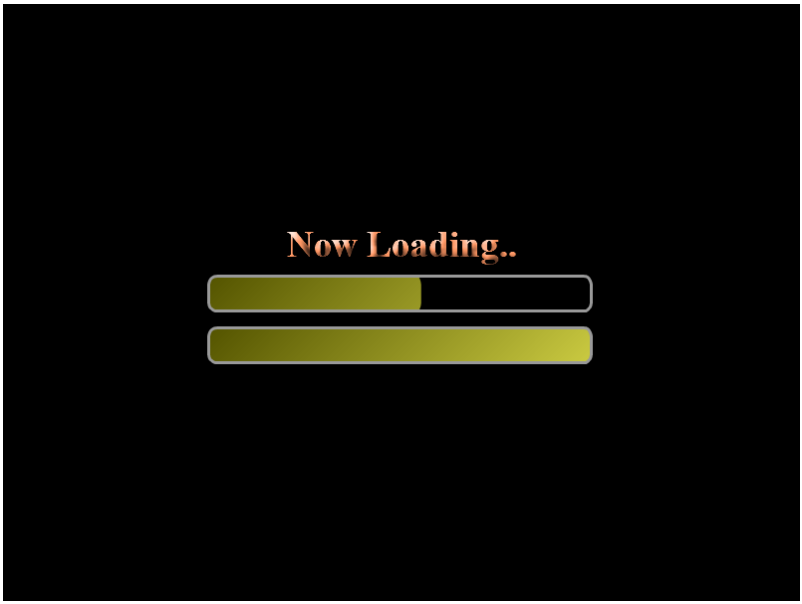
背景やキャラクターを読み込むのと同じように、『path』にファイルを指定します。そして[StartCache]コマンドで処理を開始します。

規模の大きなゲームの場合、すべてのファイルを処理するのは大変かもしれません。それにプレイ開始前に読み込み時間がかかりすぎるのも、プレイヤーにとっては好ましくありません。MB を超えるようなファイルのみを指定する、章ごとに小分けして処理するなどの工夫をしましょう。

さて[StartCache]コマンドには『wait』パラメータがあります。これを省略すると『true』が指定され、処理が完了するまで待機します。『false』を指定すると、スクリプトを進行しながらバックグラウンドで読み込みを行う「先読み」となります。

```
[StartCache  wait='false']
```

重いファイルはキャッシュ、軽いファイルは先読みなどと使い分けるのがよいでしょう。



●キャッシュしたファイル情報を削除する

AS3 版はキャッシュが保存されていれば、一度中断してもスムーズに再開することができますが、キャッシュが削除された事態を想定して（プレイヤー自身が消してしまうなど）、[Cache]コマンドで指定したファイルはゲームロード時に自動的に再キャッシュされます。

しかし「もう今後のシーンで使わないファイルまで再キャッシュされて余計な読み込み時間がかかる」というケースがあります。そんなときは「保持指定キャッシュ情報」というものを削除してしまいましょう。

```
[DelSaveCache  path='./resource/BG/sampleBG1.jpg']
[DelSaveCache  path='./resource/BGM/sampleBGM1.mp3']
```

たとえば第 2 章に移行する直前に、第 1 章で使用して以後は使用しないファイルを、[DelSaveCache]コマンドで処理をします。すると第 2 章からゲームを再開したとき、それらのファイルの再キャッシュは行われず、進行がスムーズになるわけです。

●進捗表示画面を変更する

ActionScript3.0 のスキルが必要ですが、進捗表示画面は自作のものに置き換えることができます。作り方は本書では説明しませんので、公式サイトの「スクリプト解説」から「プラグイン ー 進捗表示画面」をご覧ください。

完成した進捗表示画面は、初期設定（ini）ファイルで指定することで使用できます。[Plugin]セクションを確認してください。

```
[Plugin]

// 進捗表示画面のファイルパス
//path_Progress     = "./resource/plug-in/ProgressBar.swf"
```

コメントを削除するだけで使うことができます。

『多言語に対応させる』

●スマートフォンに最適な多言語サポート機能

LemoNovel AS3 の最大の特徴のひとつが多言語サポートです。システムが自動的に翻訳してくれるわけではありませんが、あらかじめ用意しておいた外国語のテキストファイルを、設定によって切り替えて読み込んでくれるというものです。特にスマートフォンアプリはグローバル市場で出すことができるので、数カ国語でプレイできるとなれば注目を集めやすくなります。

まずは初期設定（ini）ファイルを開きましょう。[System]セクションに以下の 2 つの項目があります。

```
// システムの言語（名称は任意）
language = "ja"

// 多言語サポート
multiLang = false
```

2 つめの『multiLang』から見てみます。『false』を『true』に書き換えてみてください。それから実行ファイルを起動してみると、何も表示されないのが確認できると思います。これは多言語サポートをオンにしたことで、スクリプトファイルの読み込み先が変わってしまったからです。

どこに変わったのかと言えば、『language』で指定しているフォルダです。ここで『ja』と指定していますが、実は使用されるシステム言語自体ではなく、フォルダ名を指定しているのです。つまり「日本語のファイルは ja フォルダに置く」という意味になるのです。

では script フォルダ内に ja という名前のフォルダを作り、その中に first.txt をはじめとするテキストファイルを入れて実行してください。今度は表示され

るはずです。ただ、[Goto]コマンドなどを使う場合、このフォルダ名を加える必要はありません。

```
[Goto path='./script/scene002.txt']
```

このように普通に記述しても、ja フォルダ内のファイルを読み込んでくれるのです。

●どの言語でも共通に読み込むファイル

言語を選択するシーンなど、どの言語でも共通に読み込みたいファイルがある場合、次のような記述で解決できます。

```
[Goto path='$COMMON:./script/scene002.txt']
```

パスの前に「$COMMON:」と追加すれば、script フォルダ直下にあるファイルを読み込ませることができます。これは初期設定（ini）ファイルでも適用可能です。

```
// 最初に実行するスクリプトのパス
firstRead= "$COMMON:script/first.txt"
```

●言語を切り替える

たとえば「言語を英語に設定しているパソコンや端末では、英語のファイルが自動的に読み込まれる」というわけではありません。言語の変更は、ゲーム内のスクリプトで行います。英語のスクリプトファイルを en フォルダに入れていると仮定します。

```
[SetSystem language='en']
[SaveSystem]
```

[SetSystem]コマンドの『language』パラメータで言語名、つまりフォルダ名を指定するのです。

言語設定はシステム情報なので[SaveSystem]で保存します。先述したとおり、システム情報は再び起動したときに自動的にロードされませんので、冒頭などに[LoadSystem]コマンドを記述するのを忘れないでください。

●プレイ途中で言語を切り替える

これまでの解説は、起動直後のタイトル画面などで言語を決定するという使い方を想定していました。しかしプレイ途中で変更した場合どうなるでしょうか。現在進行中のスクリプトには即座に反映されず、切り替え前の状態のままなのです。

これでは不自然のため、切り替え直後にもうひとつ工夫をします。

```
[SetSystem language='en']
[Reload]
[ClearHistory]
[SaveSystem]
```

スクリプトの再読込を行う[Reload]コマンドです。これにより、言語を切り替えて次のページに進んだら、英語のテキストが表示されるようになります。

なお[ClearHistory]コマンドも記述しているのは、履歴画面のメッセージが切り替え前のまま残っているからです。不自然に思うなら、このとおり消しましょう。

●メニュー項目テキスト用の翻訳テーブル

プレイ途中で言語を切り替える際に、システムメニューやコンテキストメニューを使用しているなら、これらも変更しなければなりません。

そこで初期設定（ini）ファイルを再び開きましょう。多言語サポートの項目の下にこうあります。

```
// 翻訳テーブルファイルのパス
//path_Translate    = "script/tbl_translate.txt"
```

翻訳テーブルファイルとは、メニュー項目用の翻訳テキストを記述するファイルです。コメント部分を取り除いたら新規にテキストファイルを作成し、以下の書式で記述してください。

id	matchType	ja	en
$COMMON	PERFECT	メインメニュー	Main Menu
$COMMON	PERFECT	メッセージ	Message
$COMMON	PERFECT	履歴を見る	History
$COMMON	PERFECT	セーブ	Save
$COMMON	PERFECT	ロード	Load
$COMMON	PERFECT	設定画面	Settings

1 列目が処理対象 ID です。これは翻訳を適用するメニューの ID です。個別に指定してもよいですが、コンテキストメニューでもシステムメニューでも共通で処理される『$COMMON』とすれば手間が省けます。

2 列目が一致条件です。『PERFECT』は処理対象の文字列がまったく同じなら処理されるという意味です。部分一致の『PART』という指定もでき、この場合は項目名の一致した部分のみを置換します。たとえばセーブ時にセーブ番号とシーン名を表示する形式だと……。

```
１．オープニング
２．オープニング
３．オープニング
４．オープニング
５．オープニング
```

このように番号は違っていてもシーン名は同じというケースが多々あります。翻訳テーブルファイルにシーン名を登録していても、もし『PERFECT』だと完全一致ではないので翻訳されませんが、『PART』なら部分一致しているシーン名が翻訳されるのです。

3 列目以降が各言語です。3 列目はデフォルトの言語にしてください。英語に切り替えたら「メインメニュー」が「Main Menu」となるわけです。

これら各要素をタブで区切ります。あとは必要なだけ項目と言語を加えていってください。

『プラグインを使う』

●通常のスクリプトでは不可能な機能

第１章で述べたように、そのツールに本来ない機能を別のプログラムファイルを外付けすることで実現可能にする、これがプラグインです。

プラグインの作成には、SWF ファイルを扱える環境、特に ActionScript3.0 についての知識が必要となります。かく言う筆者も、まったくそういった知識は持ち合わせておりませんが、公式サイトにはサンプルが用意されています。これを使うだけなら特別なスキルは必要ありません。

では例として「降雪の演出」を試してみましょう。公式サイトのメニュー［ダウンロード］→［プラグイン］からダウンロードしてください。

解凍するとプラグインの SWF ファイルとソースファイルが出てきます。ソースファイルは Adobe Flash Professional を用いてパブリッシュ、つまり LemoNovel 用に作成するためのものです。しかし本書ではプラグインの使い方のみを説明していきます。

●プラグインを読み込む方法

プラグインの FallSnow.swf を適当なフォルダに入れてください。今回は System フォルダ内と仮定します。そして次の記述をします。

```
[LoadBG surface='PRIMARY' path='./resource/System/FallSnow.swf']

降雪プラグインを使用しています。[p]
```

背景レイヤーに読み込みました。普通の背景画像を指定するのと何ら変わりません。しかし雪は他の背景と合わせて使う演出ですので、背景の上に読み込むという記述が好ましいです。

```
[LoadBG surface='PRIMARY' path='./resource/BG/bg03.jpg']

[LoadMovieLv level='1' kind='OTHER' path='./resource/System/FallSnow.swf' wait='NONE']

降雪プラグインを使用しています。[p]
```

ムービー再生と同じ要領で読み込んでいます。『wait』を『NONE』としないと、うまく再生されません。

しかし［LoadMovieLv］コマンドでは、背景レイヤーはもとよりメッセージレイヤーよりも上位の階層に再生してしまいます。これが不都合ならば、背景レイヤーとメッセージレイヤーの中間に配置される、キャラクターレイヤーとして再生しましょう。

```
[LoadBG surface='PRIMARY' path='./resource/BG/bg03.jpg']

[LoadChar id='0' kind='OTHER' surface='PRIMARY' path='./resource/System/FallSnow.swf']

降雪プラグインを使用しています。[p]
```

雪がメッセージレイヤーに差し掛かると後ろに隠れるのがわかると思います。

プラグインは背景レイヤー、キャラクターレイヤー、上位階層への読み込みが可能ということが理解できたでしょうか。雨や雪の場合はキャラクターレイヤーとして読み込むのが適切ですが、プラグインによって使い分けることが大切です。

公式サイトには他にもサンプルがありますので試してみてください。もっと凝ったことをしたいなら、プラグインを作れる制作仲間を募るのもよいでしょう。


降雪プラグインを使用しています。

『コラム２・本書の応用編の予定は？』

入門というタイトルなのだから、応用編の執筆の予定もあるのだろうと思われた方も、もしかしたらいるかもしれません。しかしあらかじめ申し上げておきますと、執筆の予定はありません。というのも、本書以上の内容となると、私の力量ではちょっとカバーしきれないのです。

LemoNovel AS3 における応用テクニックとは、このシステムのベースとなっている ActionScript3.0 を使いこなすことと言って差し支えないでしょう。何しろプログラミング言語なので、これを習得するには、おそらく本書を理解するより何倍もの努力（そして費用）が必要になってきます。

ですので、そんな苦労をするくらいならば本書の内容をきっちりマスターして、その範囲内で自分なりの工夫ができるようになるほうがよいと思います。それだけでも商業作品にも劣らないほどのシステムの作品を作れます。もちろん本書がすごいのではなく LemoNovel AS3 がすごいのは言うまでもありません。

なお「ちょっとややこしいけれど有用なテクニック」が本書で解説していない中にまだたくさんあります。それらは少なくとも ActionScript3.0 を覚えるほど大変ではないので、公式サイトの「スクリプト解説」や「タグリファレンス」を見ながら取り組んでみてください。

第 5 章

作品をリリースしよう

『ウェブブラウザ向けにリリース』

●LemoNovel AS3 の本来のターゲット

LemoNovel AS3 で制作した作品は、今でこそ多彩なプラットフォームでリリースすることができますが、当初からウェブブラウザ向けの Flash ゲームを作れることをウリにしていました。これがもっとも基本的なリリース方法といえるのです。

ウェブブラウザでリリースできる AS3 版は、その性質上、比較的ボリュームの小さな短編作品に向いています。読み込み時間が発生することを考えると、あまり容量の大きな作品は不向きでしょう。そして有料販売することは困難になります。

AS3 版は自分のサーバーにアップロードして、ユーザーが HTML ファイルにアクセスできるようにすれば、それでリリース完了ということになります。もうひとつ、プロジェクトフォルダを圧縮してアップロードし、ローカル環境でプレイさせることもできます。しかしこれでは各素材が丸見えになり、第 2 章で解説したセキュリティ設定をユーザーに行ってもらう手間が生じるので、おすすめはできません。

●サーバーにアップロード

アップロードは普通のウェブサイトと同じく、FTP ソフトなどを使って行います。そもそもウェブサイトを制作・公開したことがないという人は、あらかじめ習得の上で読み進めていってください。

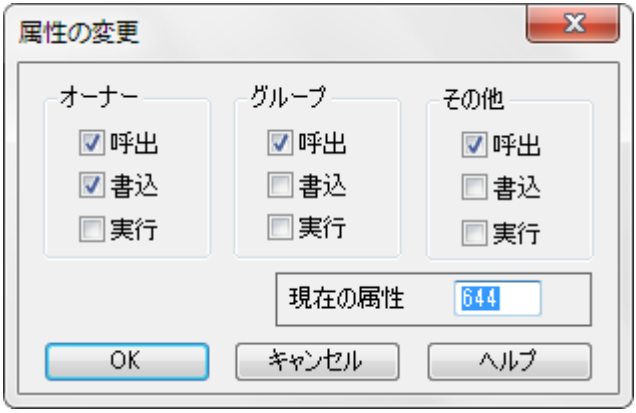
アップロードするのは、プロジェクトフォルダすべてです。フォルダの構成などはいっさいいじらないでアップロードを行ってください。なお、スクリプトファイルなど UTF-8 形式で作成されたファイルは、必ずバイナリモードで転送します。上手く動作しなかったら、ここで引っかかっている可能性があります。

アップロードが完了したら、次にパーミッションの設定をチェックします。

パーミッションとはファイルの読み書きなどの権限で、これが適切に設定されていないと、悪意のあるユーザーがファイルの改竄を行うかもしれないのです。

公式では以下の設定が推奨されているので、このとおりにしてみましょう。

	オーナー			グループ			その他			
種類	呼出	書込	実行	呼出	書込	実行	呼出	書込	実行	
swf	○	○	−	○	−	−	○	−	−	644
ini	○	○	−	○	−	−	○	−	−	644
txt	○	○	−	○	−	−	○	−	−	644
jpg	○	○	−	○	−	−	○	−	−	644
mp3	○	○	−	○	−	−	○	−	−	644
htm	○	○	−	○	−	−	○	−	−	644
他	○	○	−	○	−	−	○	−	−	644



●リリース完了

あとは HTML ファイルにアクセスすれば、ゲームをプレイできます。test.htm はゲーム画面を別ウィンドウで開く場合、start.htm は直接開く場合のファイルです。ファイル名はそのままでは味気ないので、変更しましょう。

ページデザインも自由にできるのが AS3 版の特徴です。和風、中華風、中世風など、作品内容に合わせて自由に作り込んでみてください。

『インストール形式でリリース』

●一般のパソコンソフトと同様に

AIR 版はハードディスクにインストールしてプレイします。ウェブブラウザ向けではどうしても大容量、有料販売は難しかったですが、こちらは DVD1 枚をまるまる使うくらいの容量でも問題ありません。

AIR 版のパッケージ作業は、まず Adobe AIR SDK というものをインストールするところから始めます。

【Download Adobe AIR SDK】

http://www.adobe.com/devnet/air/air-sdk-download.html

解凍場所は自由ですが、本書では C ドライブの直下に「air_sdk」というフォルダを作成し、その中に解凍したものとします。

次に公式サイトの「ダウンロード」から、制作しているのと同じ画面サイズのパッケージ作成用ファイルを手に入れます。そうしたらやはり C ドライブの直下に「pack_lemonovel」というフォルダ名にして配置します。

このフォルダの中にある bin フォルダの中身を見てください。

・icon フォルダ
・resource フォルダ
・script フォルダ
・LemoNovel_AIR.swf
・LemoNovel_AS3.ini

制作用のフォルダと似た構成になっています。このうち、LemoNovel_AIR.swf 以外を制作用のフォルダと置き換えてください。

●アプリケーション記述ファイルの編集

pack_lemonovel フォルダの中に application.xml というファイルがあります。これはアプリケーション記述ファイルといって、作品の基本的な情報を書き込むものです。テキストファイルで開いてみましょう。

いろいろと記述されていますが、編集すべき部分は下記の項目です。

```
        <id>LemoNovelAIR-800x600</id>
        <versionNumber>3.40</versionNumber>
        <filename>LemoNovelAIR_800x600</filename>

        <name>LemoNovel AIR 800x600</name>
        <description>LemoNovel AIR 作品制作用の環境</description>
        <copyright>Copyright (c) 2011-2012 LEMO.</copyright>

        <initialWindow>
                <title>LemoNovel AIR (800x600)</title>

（中略）

        <installFolder>LEMO/Develop</installFolder>
        <programMenuFolder>LEMO/Develop</programMenuFolder>
</application>
```

■id

作品固有の識別 ID です。使用できる文字は半角英数字と「.」（ピリオド）、そして「-」（ハイフン）です。一般的なのは、自分のサイトのドメインをひっくり返して、後ろに作品名をつけるというものです。私が「BAD END」という作品をリリースしたときは「net.projectlips.pcbadend」としました。

■versionNumber

作品のバージョン情報です。3 個以下の整数をピリオドで区切って指定します。初めてリリースするなら「1.00」とか「1.0.0」になるでしょう。

■filename

ユーザーが作品の起動時に実行するファイルの名前です。拡張子を含めて指定する必要はありません。また、インストールの際のフォルダ名にもなります。

■name

インストーラーで表示されるアプリケーション名を指定します。「filename」と同じ指定でよいでしょう。

■description

インストーラーで表示されるアプリケーションの説明を指定します。シンプルな説明文を記述してください。

■copyright

作品の著作権情報です。リリースした年とサークル名を記述しましょう。

■title

タイトル名です。ウィンドウのタイトルバーに表示されます。これも「filename」と同じ指定でよいでしょう。

■installFolder

作品のインストール先フォルダを指定します。たとえばこの指定を「Project Lips」、「filename」で指定したのが「BAD END」の場合、Windows 7 なら次の場所に実行ファイルやデータファイルがインストールされます。

```
C:¥Program Files (x86)¥Project Lips¥BAD END
```

よって、ここはサークル名を指定するのがわかりやすいでしょう。

■programMenuFolder

作品をインストールした際に、スタートメニューに生成されるショートカットの位置を指定します。「installFolder」と同じ指定でよいでしょう。「Project Lips」であれば[スタート] → [すべてのプログラム]→［Project Lips］と辿った先に実行ファイルのショートカットが生成されます。

これは Windows 限定の機能になります。

そのほか、アイコンの指定があります。デフォルトのアイコンが用意されていますが､サイズと形式はそのままで､オリジナルのものに置き換えましょう。

●電子証明書を作成する

次は電子証明書を作成します。「これは確かに私が作った作品です」と証明するための署名ファイルです。

第 4 章の「フォントを埋め込む」でも解説したコマンドプロンプトを起動します。そしてインストールした Adobe AIR SDK にある bin フォルダ内の adt というファイルを、下記のコマンドで実行します。

```
>cd c:¥pack_lemonovel
>c:¥air_sdk¥bin¥adt -certificate -cn NAME 1024-RSA name.pfx passwo
rd
```

実際には次の 3 ヶ所を編集してください。

■NAME

証明書の発行者を表す名前です。サークル名などがよいでしょう。

■name.pfx

署名ファイルの出力先のパスです。この例では name.pfx というファイルが

「C:¥pack_lemonovel」フォルダの中に生成されます。

■password
証明書に設定するパスワードです。

作成した電子証明書は、パッケージし直す場合や他の作品をパッケージする場合に使い回すことができますので、大事に保存しておきましょう。

●パッケージする

いよいよパッケージの準備が整いました。再びコマンドプロンプトを起動して、下記のコマンドを実行してください。

```
>cd c:¥pack_lemonovel
>c:¥air_sdk¥bin¥adt -package -storetype pkcs12 -keystore name.pfx -ts
a none air/LemoNovel_AIR_800x600.air application.xml -C bin .
```

編集箇所は 2 つです。

■name.pfx
作成した電子証明書のパスを指定します。

■air/LemoNovel_AIR_800x600.air
出力する air ファイルのパスを指定します。「C:¥pack_lemonovel」フォルダの中に air という名前のフォルダを作成しておけば、その中に LemoNovel_AIR_800x600.air というファイルが出力されることになります。このファイル名を作品名等に変更してください。

コマンドを実行すると設定したパスワードを求められますので、入力してください。このとき画面に何も表示されませんが、入力はちゃんと行われています。成功すれば air ファイルがパッケージされ、作業は完了です。

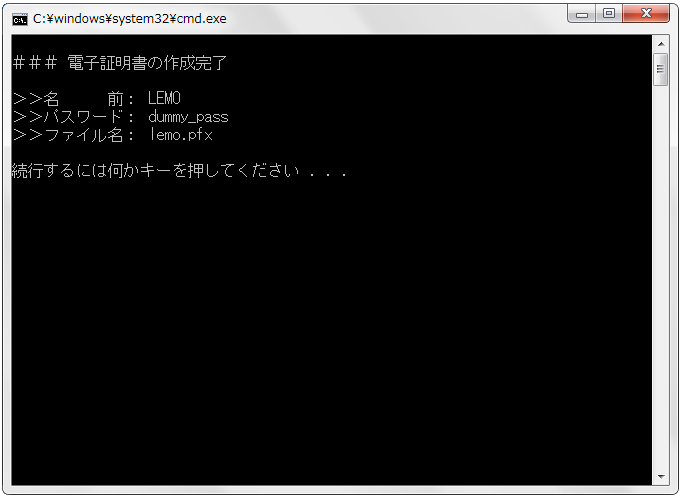
●バッチファイルを用いてパッケージ

コマンドプロンプトが面倒くさいという人は、Windows ユーザーに限り、バッチファイルを用いて電子証明書の作成からパッケージングまでを簡単に行うことができます。バッチファイルとは、コマンドプロンプトに行わせたい命令をまとめたものです。パッケージ作成用ファイルの中にある CreateCertificate.bat と PackageApplication.bat がそれです。

まず CreateCertificate.bat をテキストファイルで開いてください。

```
::$ $ $  電子証明書の情報
::－－－ 名前(任意)
set C_NAME=LEMO
::－－－ パスワード(任意)
set C_PASSWORD=dummy_pass
::－－－ 出力するファイル名(任意)
set C_OUTFILE=lemo.pfx
```

デフォルトではこのようになっているので、それぞれ編集して保存します。そうしたらファイルを実行してください。電子証明書が生成されます。



次に PackageApplication.bat を、同じくテキストファイルで開きます。編集箇所は次の 2 ヶ所です。

```
::$ $ $  電子証明書のファイル名
set  C_FILE="lemo.pfx"

（中略）

::$ $ $  出力する「.air」ファイル名
set  OUT_AIR_FILE=air/LemoNovel_AIR_800x600.air
```

編集が終わったらファイルを実行してください。CreateCertificate.bat で設定したパスワードを入力し、成功すれば air ファイルが生成されます。Windows ユーザーはこのほうが簡単なのでおすすめです。

●リソースを暗号化する

完成した air ファイルを試しにインストールしてみましょう。通してプレイして特に問題が発見されなければ、サーバーにアップしてリリースできます。

しかしインストールフォルダを覗いてみると、リソース、つまり各種素材ファイルが丸見えになってしまっています。これは air ファイルの仕様なのですが、気になる人は多いと思うので、ファイルの暗号化を行う方法を覚えましょう。

暗号化ツールは公式の「スクリプト解説」→「リソースの暗号化について」からダウンロードできます。

フィールドが 3 つありますが、入力できるのは「暗号化鍵」だけです。これはパスワードで、4 バイト以上、つまり 4 文字以上の半角英数字で入力します。復号化するときに、これが必要になります（ただし復号化ツールは公式でも頒布されていません）。

入力すると、他の 2 つのフィールドに文字列が自動的に入ります。「起動引数または ini ファイルで指定する鍵」は、初期設定（ini）ファイルの下記の項目に貼り付けて使ってください。

```
// リソースの復号鍵
//decryptKey        =
```

リソースを暗号化した場合、必ずこれを指定しなければなりません。3 つめの「プラグインから復号化を依頼する際に渡す鍵」は、本書で解説してきた知識の範囲内では必要ありませんので無視してください。

それから各種ファイルの暗号化を行います。暗号化したいファイルをそれぞれ色分けされた領域にドロップしてください。すると同じ階層に「encrypt_data」というフォルダが生成され、その中に暗号化されたファイルがあります。

フォルダごとドロップすることもできます。たとえば背景画像が入っている BG フォルダをドロップすれば、resource フォルダ内に encrypt_data フォルダが生成され、その中に暗号化されたファイルが入った BG フォルダが生成される、という流れになります。

パッケージの際に、これらの暗号化されたリソースを使ってください。テキストファイルを暗号化したなら意味不明な文字列に変換され、画像であればどんなビューアでも読み込むことができなくなります。

●自動アップデート機能

最新バージョンがリリースされると、自動的にそれをチェックしてアップデートを行えるという、Adobe AIR 特有の機能があります。

自動アップデート機能に必要なのは、最新の air ファイルと更新記述ファイルです。更新記述ファイルは新規にテキストファイルを作成した上で UTF-8 に変更し、XML 形式として保存し直してください。そして以下のように記述します。

```
<?xml version="1.0" encoding="utf-8"?>
<update xmlns="http://ns.adobe.com/air/framework/update/description/1.0
">
        <versionNumber>1.01</versionNumber>
        <url>http://www.sample.jp/sample.air</url>
        <description><![CDATA[【アップデート内容】
        ・不具合を修正しました。]]></description>
</update>
```

■versionNumber

作品の最新バージョンです。パッケージの際の値と同じにしなければエラーが出ます。

■url

最新の air ファイルの絶対パスです。

■description

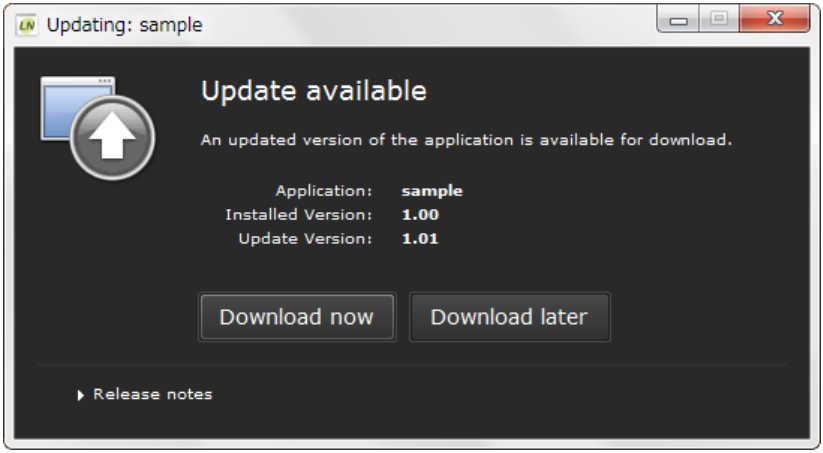
アップデート内容等を記載します。

この xml ファイルを、最新の air ファイルと一緒にサーバーにアップロードしてください。

しかし、あらかじめ最新バージョンのチェックを行うスクリプトが記述されていなければ機能しません。サンプルの first.txt に、[ChkUpdate]というコマンドが記述されているので見てみましょう。

```
// 最新バージョンのチェックを行う
[ChkUpdate chkXML='http://www.le-mo.jp/lemo/products/LemoNovel_AS3/air/update.xml']
```

このコマンドにはパラメータが複数あるのですが、必須なのは『chkXML』だけで、xml ファイルのアドレスを記述します。これで最新バージョンがアップロードされていれば、ウィンドウが出てアップデート作業を行えます。



なお、特に大容量のゲームだと、単にテキストの誤字脱字を修正したいなどの場合にはこのアップデート方法では現実的ではありません。

この場合は修正ファイルをそのまま頒布して、ユーザーがインストールフォルダにコピーするようにすればよいでしょう。もちろんファイルが暗号化されている必要があります。

『Android 向けにリリース』

●アプリケーション記述ファイルの編集

Android アプリの作成は、AIR 版とほぼ同じ流れです。異なる部分のみを解説するので、基本的なことについては前項を参照してください。

AIR Mobile 版のパッケージ作成用ファイルをダウンロードしたら、アプリケーション記述ファイルの application_apk.xml を開きます。AIR 版と共通部分が多いですが、<android>～</android>で囲まれた下記の項目は、場合によっては編集の必要があります。

■application

デフォルトで記述されている内容は、後述する APK Expansion Files を利用する場合のみ必要です。利用しない場合は、<application>～</application>ごと削除します。

■uses-permission

アプリの動作で許可が必要な権限を指定します。

機能	必要な権限
インターネットに接続する	INTERNET
スリープ不可にする	DISABLE_KEYGUARD WAKE_LOCK
外部ストレージ（SD カード等）への読み書き	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE
バイブレーション制御を行う	VIBRATE
APK Expansion Files の利用	ACCESS_NETWORK_STATE ACCESS_WIFI_STATE CHECK_LICENSE

	DISABLE_KEYGUARD INTERNET WAKE_LOCK WRITE_EXTERNAL_STORAGE
ライセンス認証をチェックする	CHECK_LICENSE

不要だと思った機能については、削除してかまいません。

そしてアイコンですが、icon フォルダの中に Android 用と iOS 用、それぞれ異なるサイズのファイルが入っています。アプリケーション記述ファイルには Android 用の指定がされているので、iOS 用は削除してください。

●電子証明書を作成する

AIR 版と同じ手順で電子証明書を作成します。ただし、AIR 版と若干異なる点があります。

```
>cd c:¥pack_lemonovel
>c:¥air_sdk¥bin¥adt -certificate -cn NAME -validityPeriod 30 1024-RS
A name.p12 password
```

■validityPeriod

AIR 版では記述しなかった電子証明書の有効期限です。これを 2033 年以降にすることが必要です。この例では現時点から 30 年間有効ということになります。

■name.p12

AIR 版では拡張子が pfx でしたが、AIR Mobile 版では p12 とします。どう違うのかというのは気にしなくてかまいません。

●パッケージする

コマンドプロンプトで、以下のように入力して実行します。

```
>cd c:¥pack_lemonovel
>c:¥air_sdk¥bin¥adt -package -target apk -storetype pkcs12 -keystore
name.p12 apk/LemoNovel_AIR_Mobile_800x480.apk application_apk.xml
 ./bin/LemoNovel_AIR_Mobile.swf bin -extdir lib/
```

これでapkフォルダを作っておけば、LemoNovel_AIR_Mobile_800x480.apkが生成されます。

●バッチファイルでのパッケージ作成

Android 版もバッチファイルでパッケージを作成できます。署名ファイル作成用の CreateCertificate.bat を開いてください。

```
::$ $ $ 電子証明書の情報
::－－－ 名前(任意)
set C_NAME=LEMO
::－－－ 有効期間(任意)
set C_VALIDITYPERIOD=23
::－－－ パスワード(任意)
set C_PASSWORD=dummy_pass
::－－－ 出力するファイル名(任意)
set C_OUTFILE=lemo.p12
```

これらデフォルトの部分を編集して保存し、実行してください。

電子証明書が作成されたら、PackageApplication_apk.bat を開きます。

```
::$ $ $ 電子証明書のファイル名
set C_FILE="lemo.p12"

（中略）
```

```
::$ $ $ 出力する「.apk」ファイル名
set OUT_APK_FILE=apk/LemoNovel_AIR_Mobile_960x640.apk
```

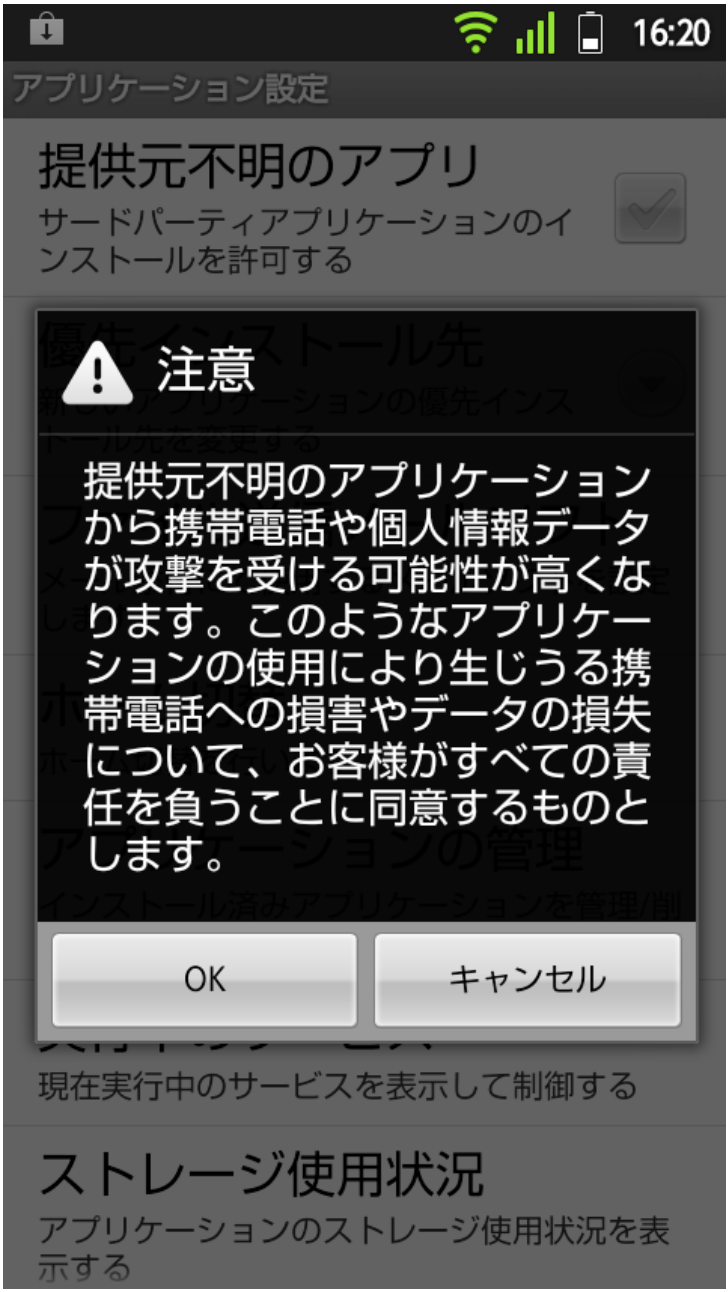
これも同様に編集して保存、実行します。パスワードを正しく入力すれば、apk ファイルが作成されます。

●端末でテストプレイをする

AS3 版も AIR 版も、開発環境とリリース環境が同じパソコンのため、テストプレイは非常に容易です。しかし AIR Mobile 版は手元の端末でテストプレイを行わなければいけません。

パッケージングが終わったら、自分のサイトにアップロードしましょう。ただし Google Play 経由でない Android アプリ、いわゆる野良アプリは初期設定のままではダウンロードできません。

設定で「提供元不明のアプリ」にチェックを入れてください（設定する画面が「アプリケーション設定」「セキュリティ」など、機種によって違いがあります）これで URL にアクセスすれば、ダウンロードできるようになります。


16:20
アプリケーション設定
提供元不明のアプリ
サードパーティアプリケーションのインストールを許可する
注意
提供元不明のアプリケーションから携帯電話や個人情報データが攻撃を受ける可能性が高くなります。このようなアプリケーションの使用により生じうる携帯電話への損害やデータの損失について、お客様がすべての責任を負うことに同意するものとします。
OK
キャンセル
現在実行中のサービスを表示して制御する
ストレージ使用状況
アプリケーションのストレージ使用状況を表示する

●Google Play に登録

テストプレイが終わり、問題ないと判断したら、Google Play にアプリを登録します。そのまま野良アプリとしてリリースしてもかまわないのですが、人目に付く可能性が段違いなので、やはり Google Play に登録するのを原則としましょう。

【Google Play Developer Console】
https://play.google.com/apps/publish/

Google Play への登録には Google アカウントと登録料 25 ドルが必要で、支払いにはクレジットカードを使います。この登録料は初回だけで、年間登録料は必要ありません。

支払いを済ませ、デベロッパープロフィールを登録したら、「新しいアプリを追加」から APK ファイルの登録をします。

アプリ情報として説明文、高解像度アイコン、スクリーンショット等が必要なので、あらかじめ用意しておきましょう。

アプリ情報の入力が終わったら、apk ファイルをアップロードします。ここで重要なのが、アップロードできるのは 50MB までという制限があることです。

これを超えるアプリについては、前編後編などに分割するか、後述する拡張ファイルを使用してください。

アップロードから 1～2 時間も経てば、パソコンや Android 端末の Google Play で検索できるようになります。なおその際に、Google Play 運営から登録完了などの連絡はありません。

あとは自サイトからアプリのアドレスに誘導して、ユーザーにプレイしてもらいましょう。QR コードを作成して、アドレスを端末で読み取らせるのもよい方法です。

●拡張ファイルをダウンロードできるようにする

LemoNovel AIR Mobile は「APK Expansion Files」という機能に対応しています。本体の apk ファイルとは別の拡張ファイルをダウンロードできるようにする機能で、最大 4GB までの拡張が可能になります。

拡張ファイルは ZIP に圧縮された「main」「patch」の 2 種類を使うことができ、各 2GB が上限です。まずは拡張ファイルに含めるべきファイル、含めるべきでないファイルを分けることから覚えてください。公式で推奨している区分を引用します。

本体パッケージ(apk)	拡張ファイル(zip)
・本体ファイル (LemoNovel_AIR_Mobile.swf) ・起動引数ファイル（bootArg) ・初期設定（ini）ファイル (LemoNovel_AS3.ini) ・外部ストレージに配置できないファイル（群） ＞埋め込みフォントデータ等	・マクロ定義ファイル ・スクリプトファイル ・その他の定義ファイル ・画像、音声などのリソース ・etc...

アプリの容量を食うのはだいたいグラフィックとサウンドでしょうから、それらは拡張ファイルにする、ただし SWF は拡張ファイルに含めないと考えればよいでしょう。

さて、拡張ファイルにまとめられているリソースは、「外部ストレージのファイル保存用のパス」に解凍される仕様になっています。つまり本体ファイルにまとめられているリソースとは、読み込み先が異なるということです。このため、スクリプト内で指定するパスの記述形式を分ける必要があります。

これまでテキストファイルや画像のパスは相対パス（本体ファイルから見てどの階層にあるか）で指定してきました。パスの頭に「./」とつけていましたが、これを省略することで、外部ストレージのファイル保存用のディレクトリを指定することができます。実例を見てみましょう。

```
// 本体ファイルの格納ディレクトリを基準とする
[LoadFont path='./ipag.swf' defName='IPAG_Normal']

// 外部ストレージのファイル保存用ディレクトリを基準とする
[LoadBG path='resource/BG/bg01.jpg']
```

パソコン上で動作させる分には、どちらの方法でも表示されます。APK Expansion Files を使用すると事前にわかっているなら、この形式を使い分けてスクリプトを記述していってください。

次に拡張ファイルに含めるリソースをフォルダにまとめ、ZIP 形式で圧縮します。フォルダ名は「main.zip」などとしてください。含めないリソースは bin フォルダの中に移しておきます。以下はフォルダ構成の一例です。

パッケージ作成用 （bin フォルダ内）	拡張ファイル作成用 （任意のフォルダ内）
ipag.swf	icon フォルダ
LemoNovel_AIR_Mobile.swf	resource フォルダ
LemoNovel_AS3.ini	script フォルダ

拡張ファイルが完成したら、初期設定（ini）ファイルを開き、APK Expanison Files の項目を編集していきます。

```
[APK_Expansion_Files]

// APK Expanison File の有効／無効[true:有効 false:無効]
// ※Android 環境以外では必ず false を指定してください
enabled             = false

// main ファイルのバージョン[-1:ファイルなし]
mainVer             = -1
// patch ファイルのバージョン[-1:ファイルなし]
patchVer = -1

// main ファイルのサイズ(byte)
mainFSize           = 0
// patch ファイルのサイズ(byte)
patchFSize          = 0

// その存在で main ファイルダウンロード処理済みと見なすファイルのパス
// (ディレクトリでも可)
mainFPath           = ""
// その存在で patch ファイルダウンロード処理済みと見なすファイルのパス
// (ディレクトリでも可)
patchFPath          = ""

// 処理完了後にダウンロードした obb ファイルを削除するか否か
// [true:する false:しない]
delObb              = true
```

■APK Expanison File の有効／無効

ここは当然「true」と書き換えます。

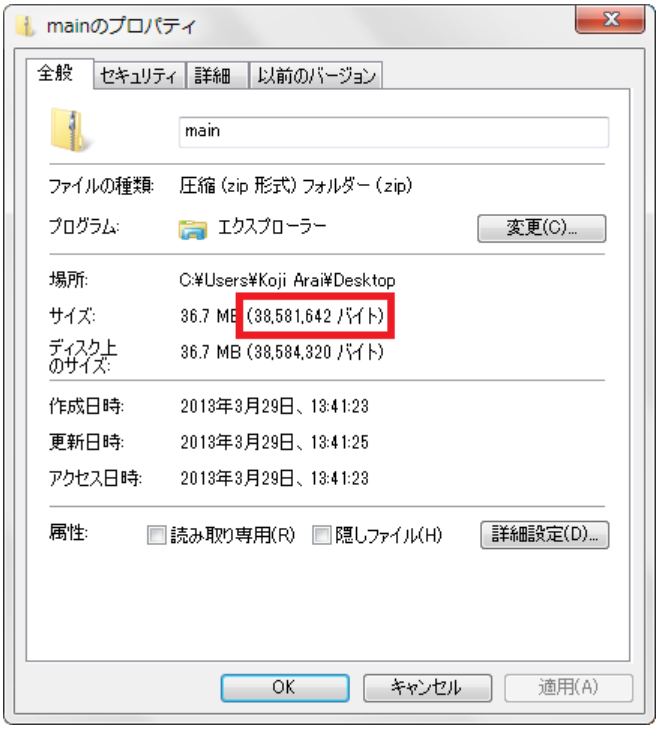
■main/patch ファイルのバージョン

拡張ファイルのバージョンを指定します。アプリケーション記述ファイルでのバージョンを 3 桁 0 埋め表記に置き換えた上で、ピリオドを外して連結します。ただし先頭のみ 0 埋めはしません。

たとえば「1.2.0」は、0 埋めすると「1.020.000」なので、連結して「1020000」となります。

■main/patch ファイルのサイズ

拡張ファイルのサイズです。これを求めるには、ZIP ファイルをプロパティで開きます。



赤で囲った部分をコピーして貼り付けてください。これが正確でないとダウンロードに失敗します。

■その存在で main/patch ファイルダウンロード処理済みと見なすファイルのパス

拡張ファイルのダウンロード処理が完了したことを証明するためのファイルを指定します。「script/first.txt」などがよいでしょう。頭に「./」をつけないことに注意してください。

■処理完了後にダウンロードした obb ファイルを削除するか否か

これは「true」のままにしましょう。「false」では外部ストレージの容量を余分に消費することになります。

最後に Android 関連の共通設定も編集します。

```
[Android_Common]

// Google Play デベロッパーの登録で付与された公開鍵
// (デベロッパーコンソールのプロフィール編集から参照可能)
// 公開鍵を必要する機能を利用しない場合は指定する必要はありません
publicKey            = 【公開鍵(Base64)】
```

デベロッパーコンソールでアプリを選択し、「サービスと API」からライセンスキーをコピーして貼り付けてください。

これですべて完了ですので、先述した方法でパッケージングしてください。そうしたら apk ファイル、main ファイル、patch ファイルの順にアップロードして、正しく動作するか端末で確認しましょう（本書執筆時点では、拡張ファイルのアップロードは旧デザインのデベロッパーコンソールでのみ行えます)。

充分にテストして OK となったら、いよいよ公開です。

『iOS 向けにリリース』

●iOS アプリのリリースに必要なもの

LemoNovel AS3 作品の開発は Windows でも Mac でも行えるのですが、リリースするには Mac 本体が必要になります。OS のバージョンは必要なツールの関係で「Mac OS X 10.7 Lion」以降ですので、それ以前を使っているならアップグレードしておきましょう。そして「iOS Developer Program」というライセンスを取得しなければなりません。年間 99 ドルと、Android に比べるとかなり高めに設定されています。

まずは Apple Developer に Apple ID を登録して、ライセンスを購入しましょう。このとき、登録内容はすべて英語で行うというのがポイントです。

【Apple Developer】
https://developer.apple.com/jp/

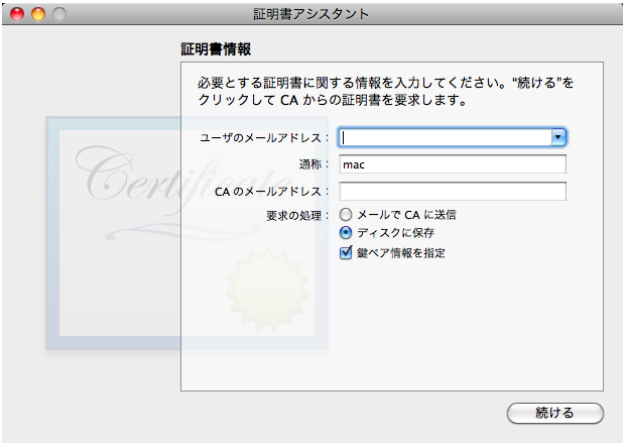
すでに Apple ユーザーで Apple ID を持っている人も多いと思いますが、アカウント情報に日本語が含まれていると、問題が発生することがあります。そのため､面倒でも半角英数字のみの新しい Apple ID を作りなおしてください。

●開発用証明書の取得

最初に行うのは、開発用証明書の取得です。これを AIR 版や Android 版でも使った電子証明書に変換するのです。

取得は Mac で行います。ユーティリティーの「キーチェーンアクセス」を起動してください。まず環境設定の「証明書」タブで「オンライン証明書状況プロトコル（OCSP)」と「証明書失効リスト（CRL)」が「切」になっているかどうか確認してください。

次にメニューから「証明書アシスタント」→「認証局に証明書を要求」と選択します。



■ユーザのメールアドレス

Apple Developer に登録しているメールアドレスを入力します。

■通称

Apple Developer に登録している氏名を入力します。

■CA のメールアドレス

未入力のままにしてください。

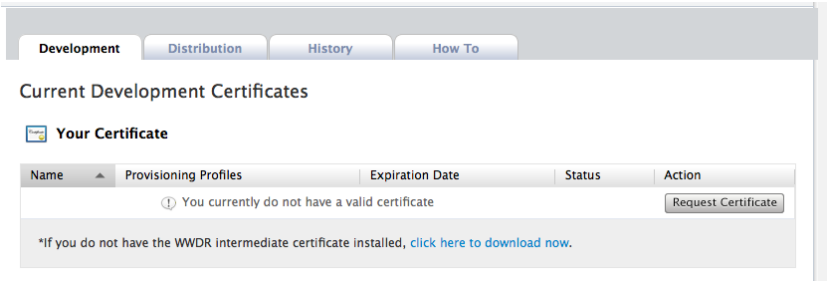
■要求の処理

「ディスクに保存」「鍵ペア情報を指定」にチェックを入れます。

ファイルは「CertificateSigningRequest.certSigningRequest」という長い名前ですが、そのままで任意の場所に保存してください。

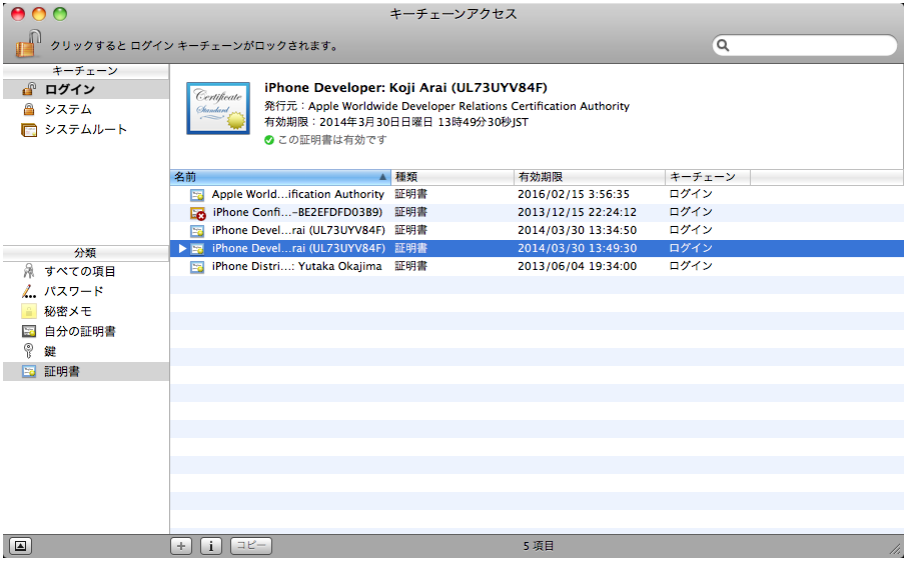
さらに鍵ペア情報という項目を入力します。内容は理解できなくてかまいませんので、「鍵のサイズ」を「2048 ビット」、「アルゴリズム」を「RSA」としてください。

いわば開発用証明書を取得するための申請書であるこのファイルを、窓口である iOS Dev Center にアップロードします。右側のメニューにある「iOS Provisioning Portal」にアクセスしてください。その中の「Certificates」から「Request Certificate」ボタンを選択します。



CertificateSigningRequest.certSigningRequest を送信すると元の画面に戻ります。ここでページを更新すると、先程はなかったダウンロードボタンが表示されています。これをクリックすると証明書ファイルの「ios_development.cer」が保存されます。

次にこのファイルをダブルクリックしてください。再びキーチェーンアクセスが起動します。証明書がインストールされるので、「分類」にある「証明書」を表示してください。それを選択した状態でメニューの「ファイル」→「書き出す」を選択します。p12 のフォーマットで保存できます。ファイル名はサークル名などでよいでしょう。最後に書き出しに必要なパスワードを入力し、保存してください。



●テストプレイ用の端末を登録

ここでテストプレイのための iOS 端末を登録しておきましょう。引き続き Mac で行います。

端末をパソコンに接続したら、iTunes を起動してください。デバイスを選択すると端末のシリアル番号が表示されますが、これをクリックすると「識別子（UDID)」というものに変わります。これを control キー+クリックからコピーしてください。

この UDID を iOS Dev Center へ登録します。「iOS Provisioning Portal」の「Devices」に移動し、「Add Devices」をクリックしてください。デバイス名と UDID を入力すれば登録が完了します。デバイス名は「MY iPhone5」などがよいでしょう。

●アプリの ID（App ID）を設定する

リリースする作品の ID、App ID（Apple ID とは別物）を設定します。「iOS Provisioning Portal」の「App IDs」に移動してください。入力欄が 2 種類あります。

なお、App ID は一度登録すると消すことができません。くれぐれもテスト感覚で登録しないようにしてください。

■Description

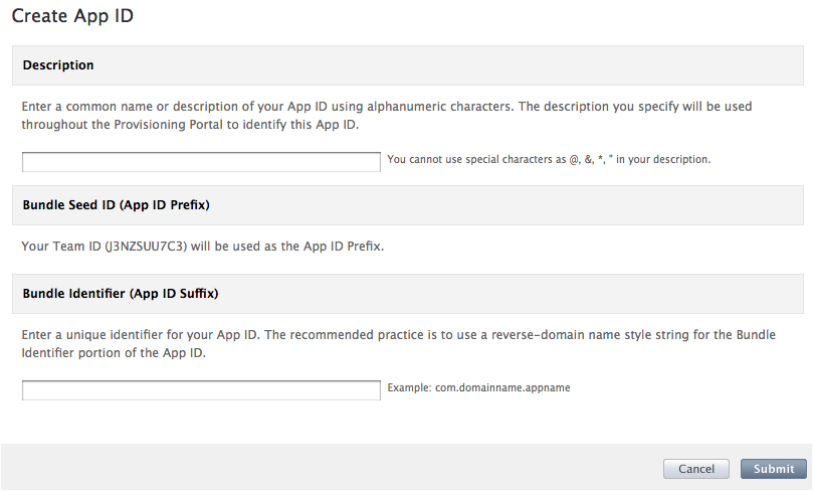
アプリの説明文です。シンプルにタイトルを入力してもよいでしょう。

■Bundle Identifier（App ID Suffix）

アプリの ID です。これをあとでアプリケーション記述ファイルに指定することになります。AIR 版や Android 版と同様の逆ドメイン形式（例：net.projectlips.iosbadend）で入力してください。

ここで「*」（アスタリスク）のみを入力するという特殊な方法があります。こうすると App Store への登録ができませんが、特定の ID を使わないさまざまなアプリをパッケージングすることができます。テスト的なアプリを作って端末で操作してみたいというケースに有用です。

「*」とした場合は、Description に入力するのは特定のタイトル名ではなく、自分の名前やサークル名にすることになります。



●プロビジョニングプロファイルの取得

プロビジョニングプロファイルとは、iOS デバイス、証明書、アプリケーション ID を関連づけたファイルのことで、パッケージ作業の際に証明書とセットで使用することになります。「iOS Provisioning Portal」の「Provisioning」に移動してください。そして「New Profile」ボタンをクリックします。

注意しなければならないのは、プロビジョニングプロファイルは開発用とリリース用の 2 種類があることです。「Development」タブで作成するのが開発用、「Distribution」タブで作成するのがリリース用です。

入力欄は以下の 4 つです。

■Profile Name

任意の名前を入力します。開発用とリリース用で区別する必要があります。私が「名作サウンドノベルシリーズ・イワンの馬鹿」というアプリを制作したときは以下のようにしました。

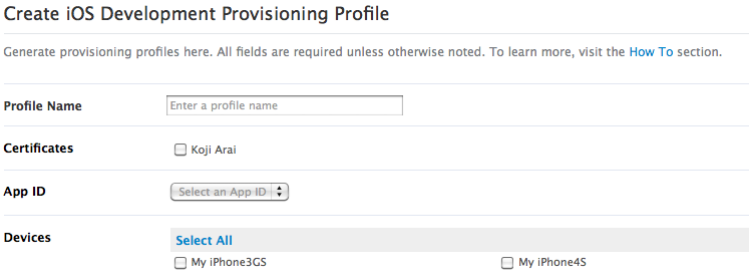
開発用：Meisaku Iwan
リリース用：Meisaku Iwan Apps

■Certificates
作成した証明書が表示されているのでチェックします。

■App ID
登録した App ID を選択します。

■Devices
どの端末でテストプレイをするかを選択します。リリース用ではこの項目はありません。

これで拡張子が mobileprovision のプロビジョニングプロファイルが作成されるので、保存してください。



●アプリケーション記述ファイルの編集

長い下準備が終わり、いよいよパッケージ作業に取りかかります。パッケージ作成用ファイルに含まれている application_ipa.xml を開いてください。

基本部分は Android 版と共通ですが、「id」は iOS Provisioning Portal で発行したものを指定します。もし「*」としていたら、ここは任意の文字列を入れてかまいません。また「filename」は端末のホーム画面に表示される名前で、iOS の場合、全角 6 文字までしか入りません（それを超えると切り取られる）。

見栄えをよくしたいなら、この範囲内に収まる略称を入れましょう。

そして<iPhone>～</iPhone>に囲まれているのが iOS 版独自の部分です。

```
<iPhone>
        <InfoAdditions><![CDATA[
                <key>UIDeviceFamily</key>
                <array>
                        <string>1</string>
                        <string>2</string>
                </array>
                <key>UIApplicationExitsOnSuspend</key><true />
                ]]></InfoAdditions>
        <requestedDisplayResolution>high</requestedDisplayResolution
>
</iPhone>
```

■UIDeviceFamily

サポートするデバイスを定義します。1 が iPhone/iPod touch、2 が iPad です。

■UIApplicationExitsOnSuspend

ホーム画面に戻るなどして画面を切り替えたとき、アプリを終了するか否かです。「true」にすると終了、「false」にするとバックグラウンドで動作し続けます。

デフォルトでは最低限の記述だけですが、特に編集する可能性があるのは以上の 2 種類になります。

●パッケージする

Mac で作成した P12 ファイルとプロビジョニングプロファイルを用意し、Android 版と同じ要領で作成していきます。

なお、Windows でパッケージ作業をする場合、Dropbox などのクラウドサービスを利用すれば、Mac との間でファイルの行き来が簡単にできます。

【Dropbox】

https://www.dropbox.com/

コマンドプロンプトで、以下のように入力して実行します。Android 版とは若干異なる箇所があります。

```
>cd c:¥pack_lemonovel
>c:¥air_sdk¥bin¥adt -package -target ipa-app-store -provisioning-profile test.mobileprovision -storetype pkcs12 -keystore name.p12 -storepass dummy_pass ipa/LemoNovel_AIR_Mobile_960x640.ipa application_ipa.xml ./bin/LemoNovel_AIR_Mobile.swf bin -extdir lib/
```

■-target ipa-app-store

最終的に App Store にリリースするバージョンを作成するならこの記述です。まだテスト段階というときは「-target ipa-test」とすると迅速に作成できます。

■-storepass dummy_pass

電子証明書のパスの指定の直後に、証明書のパスワードを記述します。「dummy_pass」を自分の決めたパスワードに置き換えてください。

■-provisioning-profile test.mobileprovision

プロビジョニングプロファイルのパスを指定します。証明書と同じ場所にコピーしておきましょう。

これで ipa フォルダを作っておけば、LemoNovel_AIR_Mobile_960x640.ipa が生成されます。

ところで AIR Mobile 版のベースとなっている AIR for iOS は、現在のところサイレントスイッチ（マナーモード）に対応していません。スイッチを切り替えても音が鳴ったままになるのです。したがって、リリースの際にマナーモード非対応とアナウンスする必要があります。

実は外部のフリーツールを使えば対応させることもできるのですが、少々複雑な手順のために本書では紹介しません。公式サイトではその手順が掲載されていますので、興味のある方は見てみましょう。

●バッチファイルでのパッケージ作成

iOS 版もバッチファイルでパッケージを作成できますが、パッケージを行う PackageApplication_ipa.bat の記述は Android 版と異なる部分があります。

```
::$ $ $  プロビジョニングプロファイルのファイル名
set  P_FILE="lemo.mobileprovision"

（中略）
echo 「.ipa」ファイルの作成を開始
call %PATH_AIRSDK%bin¥adt -package -target ipa-app-store...
```

プロビジョニングプロファイル名と、必要に応じてリリース形態を編集してパッケージしてください。

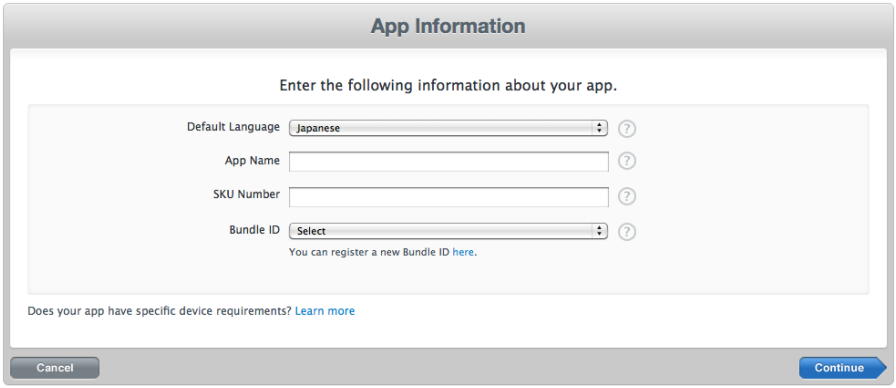
●端末でテストプレイをする

開発用のプロビジョニングプロファイルで ipa ファイルを作成したら、Mac の iTunes に入れましょう。ファイルをダブルクリックするだけで追加されます。そうしたらデバイスに表示されている端末にドラッグ＆ドロップすれば、インストールが開始します。

●App Store に登録する

テストプレイが終わったら、App Store への登録作業に入ります。iOS Dev Center の「iTunes Connect」にアクセスしてください。

ここの「Manage Your Application」から「Add New App」で登録画面に移動します。



項目が 4 つあります。

■Default Language

デフォルトの言語です。「Japanese」となっているはずなので、そのままにしておきます。

■App Name

アプリ名です。

■SKU Number

管理番号です。サークル名+ナンバリングなどが考えられます。

■Bundle ID

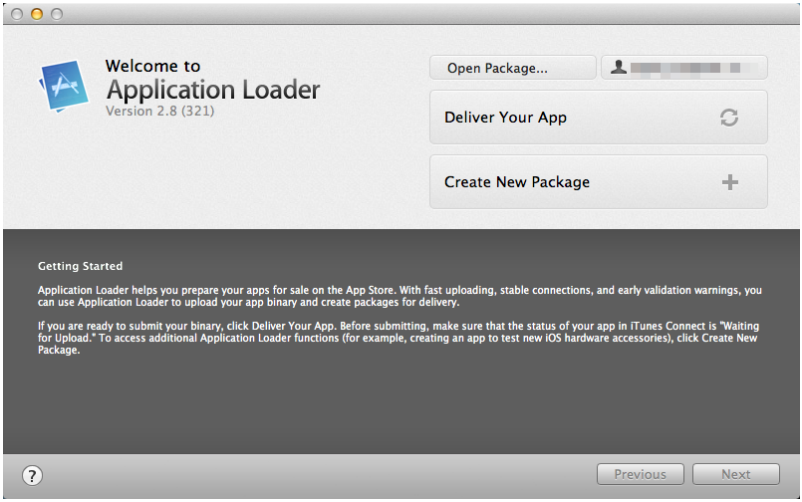
iOS Provisioning Portal で発行した App ID を指定します。

次の画面でアプリの情報やスクリーンショットなどを登録していきます。ここは特に難しい部分はないでしょう。

登録作業が終わると「Waiting For Upload」というステータスになります。あとはアプリをアップロードするだけです。アップロードに使うのは Xcode というツールに付属している Application Loader です。

いったん iOS Dev Center に戻って「Downloads」へと移動してください。Mac App Store から Xcode の最新版をダウンロードします（本書執筆時点で 4.6.1）。これが Mac OS X 10.7 Lion 以降でないと動作しません。

Application Loader は Xcode メニューの「Open Developer Tool」から起動できます。



ここで「Deliver Your App」をクリックします。「Waiting For Upload」状態のアプリが表示されているので、選択。あとは指示に従うだけでアップロードができます。すると「In Review」というステータス、つまり審査待ちになります。

審査期間はばらつきがありますが、おおむね 1 週間程度です。無事にこれを通過すればステータスが「Ready for Sale」になり、App Store でリリースが開始されます。

【著者略歴】

アライコウ

1981 年生まれ。大学卒業後、ライターとして活動開始。
パソコン系の雑誌への寄稿や、ゲームシナリオ執筆などを行っている。
著書に『XNovel でつくる iPhone ノベルゲーム』『もしものための iPhone サバイバル術』がある。

LemoNovel AS3 ゲーム制作入門

発行日：2013 年 4 月 14 日　第 1 刷発行
著者：アライコウ
カバーイラスト：瑞月シノ
発売元：同人ゲーム支援 net
http://doujingame-shien.net/
連絡先：info@doujingame-shien.net