UJML はじめの一歩

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

はじめに 5
はじめての UJML 5
１．UJML の基礎概念 6
UIEngine について 6
UJML は XML ベースの開発言語 7
ユーザーインタフェースの開発について 8
開発環境の準備 9
２．文字列の表示　Hello World 10
<label>要素を使う 10
文字列の装飾 12
３．ステート変数：基礎編 15
ステート変数の使用 15
ステート変数の概念 17
ステート変数の値の変化 18
ステート変数の扱い 18
ステート変数と<transition>要素 19
４．スクリプト 22
簡単なスクリプト 22
変数 23
変数の宣言 23
型 24
ステート変数 24
ステート変数の宣言 25
型 25
変数及びステート変数の配列 26
静的な配列 26
動的な配列 27
コメント 27
複数行コメント 27
<script>内での 1 行コメント 28
<script>内での複数行コメント 28
コメントタグ 28
定数　ENTITY 29
予約語 31
変数の要素値への使用 31

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

条件判断　if........32
比較演算子........33
繰り返し　while,for........34
５．ステート変数：応用編........36
ステート変数と delay と script........36
<transition>要素........38
ステート変数とヴィジュアル要素の描画........39
６．関数........41
関数の基礎........41
組み込み関数を使う........43
_strcat()........43
_srand()........43
_msec()........43
_getIntProperty()........44
_text_width()........44
_text_height()........44
_unload()........45
_clear_state()........45
ランダム値を作成する関数........45
10 桁のランダム値を作成する関数........46
７．ヴィジュアル要素の表示（<box>要素）........49
単純な長方形の描画........49
ヴィジュアル要素の入れ子構造........51
Z-Order の話........52
８．ヴィジュアル要素と変数........54
<box>内に<label>を配置........54
長方形のセンタリング........57
UJML における初期化の定石........59
画面サイズの取得........59
文字列サイズの取得........59
スタートボタンの作成........60
９．イベントの処理........64
キー入力によるイベントの処理........64
ファンクションキーによるイベントの処理........67
数値ボタンによる入力イベントをチェックする話、配列使う........68
ステート変数の変化によるイベントの処理........71

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

ゲームの完成........73
１０．画像の表示........83
簡単な画像の表示........83
リソースの利用........84
画像によるおめでとうメッセージの表示........88
付録........98
開発者向けティップス、注意事項........98
開発環境（デバッガ等）の注意事項........100
より高度なアプリケーション設計に向けて........100
パーティションについて........100
_run()関数........100
_link()関数........101
メモリリソースを使用するパーティション........102
ディスプレイテンプレートの活用........102
オブジェクト指向によるアプリケーション設計........103

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

はじめに

はじめての UJML

本ドキュメントは UJML による UIEngine アプリケーション開発初心者を対象にしています。但し、プログラミング初心者のための基本的なプログラミング解説はしておりませんので、C や JAVA といった他の言語でのプログラミングの基礎的な概念を理解した前提の構成となっております。プログラミング初心者の方は必要であれば書籍や研修等でプログラミングについてまず学んでください。

また、本ドキュメントで解説する UJML は初歩的な内容です。複雑なアプリケーションやゲームを開発する際には、より高度な設計が求められます。より高度な設計については本ドキュメントでは触れません。

本ドキュメントでは簡単なゲームを作りながら UJML の基本を学びます。一つのプログラムを重要な項目に分けて解説し、最終的には一つの動くプログラムを完成します。ゲームは「10 桁の数字を数秒間で記憶した後、間違えないように解答する」という簡単なものです。簡単なゲームですが、抑えるべき UJML の要素を網羅的に紹介しています。

本ドキュメントの内容と関係なく UJML のバージョンアップにより言語仕様が変更される可能性があります。本ドキュメントは UJML2.1 を対象としており、それ以降の言語仕様での動作を保証するものではありません。

また、本ドキュメントの作成にあたっては正確な記述に努めましたが、本ドキュメントの内容に対して何らかの保証をするものではありません。また、当社の故意又は重過失の場合を除き、内容やサンプルに基づくいかなる運用結果に関しても一切の責任を負いません。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

１．UJML の基礎概念

本章では、UJML を学ぶ前に理解の前提となる UIEngine の仕組と UJML の概念を簡単に解説していきます。

UIEngine について

UIEngine は小型デバイスから大型デバイスまで、全てのデバイスに UI（ユーザーインタフェース）を効率的に提供できるように設計されています。何だか難しい話ですが、UJML でアプリケーションを開発してユーザーに使ってもらうまでの流れを簡単に解説しましょう。

まず、開発者（あなた）が携帯電話用のアプリケーションを作りユーザーが使うまでの流れです。ここでは、仮にパズルゲームを例にします。

１．あなたがパズルゲームの UJML を書きます。
２．あなたが UJML をコンパイルし、バイトコードを作成します。
３．あなたがインターネットに UIEPlayer（携帯電話版）とバイトコードを配置します。
４．ユーザーが携帯電話からインターネットにアクセスします。
５．ユーザーがパズルゲームをダウンロードします。（このとき実は UIEPlayer がダウンロードされます。）
６．ユーザーがパズルゲームを実行します。（このとき実行されるのは UIEPlayer です。UIEPlayer がバイトコードをダウンロードし、パズルゲームを実行しています。）

さて、突然出てきた UIEPlayer とは何でしょうか？具体的に説明する前に、ユーザーが同じパズルゲームを携帯電話ではなく PDA で使うまでの流れを紹介しましょう。

１．あなたがインターネットに UIEPlayer（PDA 版）とバイトコードを配置します。
２．ユーザーが PDA からインターネットにアクセスします。
３．ユーザーがパズルゲームをダウンロードします。（このとき実は UIEPlayer がダウンロードされます。）
４．ユーザーがパズルゲームを実行します。（このとき実行されるのは UIEPlayer です。UIEPlayer がバイトコードをダウンロードし、パズルゲームを実行しています。）

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

どうでしょうか？PDA にパズルゲームを提供する際にはもう、UJML を書く必要がありません。これが UIEngine を使うことでアプリケーションを効率的に提供できる肝となっています。

UIEPlayer は各種デバイス毎に UIEvolution 社が提供・サポートしています。つまり、開発者は一度作成したアプリケーション（ソースコード）をデバイス毎に書き直す必要がありません。これまでは、i モード携帯には DOJA で、midp 対応端末には JAVA MIDP でと、デバイス毎に異なる言語で書き直す必要がありました。この手間を吸収できるのが UIEngine の特長の一つです。

アプリケーションのバイトコードとデバイスに合った UIEPlayer を組み合わせて提供すれば全てのデバイスで同じアプリケーションを動作させることができます。UIEPlayer はバイトコードに変換されたアプリケーションを実行するまさにプレイヤーです。

UIEngine にはアプリケーションを効率的に提供する機能が他にもたくさんありますが、長くなってしまうのでこの辺で。

ワンポイント：アプリケーションはバイトコードとデバイスに合った UIEPlayer を組み合わせて提供します。

UJML は XML ベースの開発言語

UJML は XML ベースの開発言語です。XML とは何でしょうか？ここでは XML について簡単に解説します。まず、XML は HTML のようにタグベースで記述するマークアップ言語です。例えば、画面に「HelloWorld!」を表示する UJML のソースコードはこのようになります。

helloworld.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <display>
            <label>
                <text>Hello World!</text>
            </label>
        </display>
    </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

もちろん、他の C や JAVA といったプログラミング言語と同じように、if,while,for といった制御用のステートメントを持つスクリプトを記述することも可能です。その場合には<script>要素の中に書くことになります。スクリプトは ECMA スクリプト（JavaScript）のサブセットです。もし C や JAVA、JavaScript に触れたことがあれば理解も早いでしょう。

ワンポイント：UJML は XML ベースの開発言語ですが、他のプログラム言語と同じように制御用ステートメントを持つスクリプトを記述することができます。

本ドキュメントでは UJML を解説する上で、XML の呼称を使用しています。ここで呼称について簡単に解説します。この後の具体的な解説中に何度も出てきますので早めに覚えましょう。

```
<ujml>
    <application>
        <display>
            <label>
                <text>Hello World!</text>
```

これは先ほどの helloworld.ujml を抜き出したものです。タグで表記された”<application>”は<application>要素（アプリケーション要素）と呼びます。また、<application>要素に対して、<ujml>要素を**親要素**、<display>要素を**子要素**と呼びます。

<text>要素にタグで囲まれた文字列「Hello World!」がありますが、これは**<text>要素の値**、または**要素値**と呼びます。helloworld.ujml 内にはありませんが、

```
<function name="processInput" type="void">
```

のように記述することがあります。name を<function>要素の**属性**と呼び、ダブルクォーテーションで囲まれた”processInput”を**属性値**と呼びます。

各呼称をまとめると、次のようになります。

```
<要素 属性="属性値" >要素値</要素>
```

ユーザーインタフェースの開発について

UIEngine はその名前が示すとおり、UI（ユーザーインタフェース）の開発に適した言語

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

です。また、多くのデバイスに対応するためにミニマリストな設計に基づくため、必要最低限のことに対応しています。

一つのアプリケーションを多くのデバイスへ対応させるとき、開発者がまず意識することの一つに画面サイズがあるでしょう。開発者は異なる画面サイズでも適切な表示が出来るように考慮して設計する必要があります。

そのために、UJML では画面サイズ等のデバイス情報を取得する組み込み関数が提供されています。開発者はデバイスの情報をもとにダイナミックに最適な表示ができるようにアプリケーションを設計する必要があります。開発者としてそれらの仕組を使ったアプリケーション開発になれる必要があるでしょう。

開発環境の準備

UJML アプリケーションを開発する前にまず、開発環境を準備します。UIEvolution 社より UJML2.1 の開発環境として UIEDeveloper が提供されています。この開発環境には UJML ソースコードをバイトコードにコンパイルするコンパイラが含まれています。SDK は 1.5 と 2.1 二つのバージョンがありますが SDK2.1 をインストールしてください。

SDK のインストールは

http://developer.uievolution.com/developer/download.php

からダウンロードしておこないます。

SDK の対応 OS は Windows2000、WindowsXP および、Mac OS 10.4 以降となっています。インストールには JAVA の SDK1.4.2 以降のものが必要です。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

2．文字列の表示　Hello World

本章では、文字列表示方法の紹介を通し、UJML の基本的な書き方を解説していきます。

<label>要素を使う

UJML では文字列の表示には<label>要素を使用します。<label>要素に対し、色、フォントサイズ、フォント種類を指定することで文字列を装飾することも可能です。

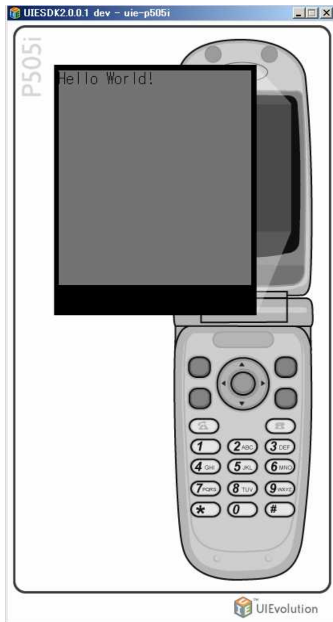
まず、単純に Hello World!と表示してみましょう。以下のソースコードを SDK のエディタで記述し実行します。

実行方法について：Package Explorer 内で実行する ujml ファイルを選択し、右クリックしたメニューから「Run As」＞「UJML Application」として実行するか、右クリックしたメニューから「Set As Main」を選択して、プロジェクトのメインファイルとして設定した後、ツールバーにある実行ボタンをクリックして実行します。

helloworld.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <display>
            <label>
                <text>Hello World!</text>
            </label>
        </display>
    </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止



いかがでしょうか？

```
            <label>
                <text>Hello World!</text>
            </label>
```

にある<text>要素の値を変更すると表示される文字列も変わります。このように UJML では文字列を表示する際に全て HTML のようにタグで記述することができます。

XML ではタグで表記されたものを要素と呼びます。本ドキュメントでは<label>要素のように表現しています。

それでは、この helloworld.ujml を簡単に解説します。まず、最初の三行は XML ドキュメントを書くときの決まりごとで、いまは「必ず書かなければいけないもの」として覚えてください。

次に、<ujml>要素と<application>要素は UJML アプリケーションを書く際に必ず記述します。これに対応して最後に</ujml>と</application>を忘れないようにしましょう。

<display>要素は画面上に何かを表示するときに使います。<display>要素の子供の要素（以下、子要素）が画面に表示されます。ここでは<label>要素が一つあります。

<label>要素は画面上に文字列を表示するときに使います。<label>要素は一行しか表示できませんので、複数行表示したいときには<multi-label>要素を使用します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

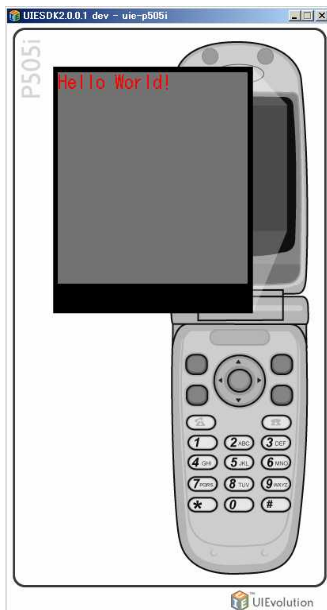
本ドキュメントでは<multi-label>要素の解説はしておりません。

文字列の装飾

では、次に、helloworld.ujml の文字に赤色をつけ、フォントサイズを大きくしてみましょう。

helloworldcolor.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <display>
            <label>
                <text>Hello World!</text>
                <fg>&_COLOR_RED;</fg>
                <size>&_FONT_SIZE_LARGE;</size>
            </label>
        </display>
    </application>
</ujml>
```



このようにして、<label>要素の子要素に<fg>要素や<size>要素を指定することで文字列の装飾を行うことができます。<fg>要素は文字の色を、<size>要素は文字のサイズを指定します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

ところで、<fg>要素や<size>要素に指定されている
&_COLOR_RED;
&_FONT_SIZE_LARGE;
が何か気になったと思います。後々説明しますが、これは UJML の定数です。UJML では色やサイズを数値で指定することもできますが、その代わりに、このような意味を持った定数名を指定できます。

また、<label>要素には次の子要素を指定することができます。特に指定しない場合には規定値（デフォルト）に基づき表示されます。

要素名	目的	規定値
text	表示する文字列	なし（省略不可）
x	X 軸方向の位置（単位：ピクセル）	0
y	Y 軸方向の位置（単位：ピクセル）	0
fg	テキストの色	黒
bg	背景色	透明
size	フォントサイズ	中
style	フォントスタイル	標準
face	字体	標準
event	イベントを（キー入力等）関連付ける	なし
ヴィジュアル要素	<box>等の子要素	なし

helloworldcolor.ujml では<fg>要素、<size>要素を指定しました。helloworld.ujml では指定していないので、規定値に基づいて画面に表示され、テキストの色は黒、フォントサイズは中となります。

helloworldcolor.ujml では<fg>要素に定数を指定しましたが、16 進数で色を指定する方法があります。これは HTML で赤を FF0000 と指定するのに極めてよく似ています。

UJML で仮に赤を指定する場合には

```
<fg>0xFFFF0000</fg>
```

と書きます。最初の 0x はこれから 16 進数を書きますという宣言で、その後に続く 8 文字で色を指定します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

FF	FF	00	00
透明度合	赤	緑	青

最初の二文字で透明度合を指定します。FF なら不透明、00 なら透明となります。00 の場合は色が表示されません。その次から二文字ずつ赤、緑、青の色を指定します。例えば 0x8800FF00 とすると、背景が透けて見える緑色になります。

注意：半透明になるかどうかは UJML アプリケーションが実行されるデバイスに依存します。半透明を描画する機能があるデバイスの場合は半透明で描画しますが、そうでない場合は、透明度が 1 以上のものを全て不透明として描画します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

３．ステート変数：基礎編

本章では「ステート変数」を解説していきます。UJML を扱うには、まず、この言葉を覚えなければなりません。ステート変数はその名の通りアプリケーションの状態を管理する、あるいは状態を表す変数です。

他のプログラミング言語には出てこない特殊な変数ですが、普通の変数と同じように取り扱うことができます。難しいことはありません。何度か経験することですぐなれることがでるでしょう。

ステート変数の使用

下のソースコードを見てください。この state.ujml は helloworld.ujml と同じように画面上に文字「Hello World!」を表示します。

state.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sHelloWorld" type="boolean"/>
        </state-variables>
        <script>
            sHelloWorld = true;
        </script>
        <states>
            <state var="sHelloWorld">
                <transition value="true">
                    <display>
                        <label>
                            <text>Hello World!</text>
                        </label>
                    </display>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

まず、ブーリアン型のステート変数 sHelloWorld を宣言しています。ステート変数の宣言

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

には<state-var>要素を使います。ここでは、このステート変数 sHelloWorld を文字列「HelloWorld」の表示・非表示を切り替えるために使用しています。

ステート変数 sHelloWorld が<script>要素内で true に設定されています。そして、<states>要素内に sHelloWrold が true に設定されたときの振る舞いが記述されています。先ほど HelloWorld を表示するときにあった<display>要素と同じものですね。

さらに<states>要素内を詳しく解説します。まず、

```
            <state var="sHelloWorld">
```

は、この要素内でステート変数 sHelloWorld に値が設定されたときに、アプリケーションが何を表示するのか、どのような動作を行うのかといったアプリケーションの振る舞いを記述することを示しています。

```
                <transition value="true">
```

は、ステート変数 sHelloWorld の値が true に設定されたときのアプリケーションの振る舞いを記述することを示しています。ステート変数 sHelloWorld の値が true 以外に設定された場合にはこの要素以下のものは無視されます。

このように、ステート変数に設定された値に対応するアプリケーションの振る舞いを具体的に記述することができます。

内部的にはステート変数の変化はイベントとして処理されます。ステート変数が変化する毎にイベントが発生し、ステート変数に設定された値に対応する<transition>要素に記述された内容が処理されます。

ステート変数を理解しなければ UJML アプリケーションを開発できないといっても言い過ぎでは無いでしょう。この後の章でもステート変数の解説がありますが、しっかりと理解していきましょう。

また、state.ujml では<script>要素内にスクリプトがあり、sHelloWorld に true 値を設定しています。

```
        <script>
            sHelloWorld = true;
        </script>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

UJML が扱うスクリプトについては後ほど解説しますが、まずここでは、UJML では<script>要素内に他の言語と同じようにスクリプトを書くことができると覚えてください。

<application>要素の子要素として記述される<script>要素をデフォルトスクリプトと呼びます。デフォルトスクリプトはアプリケーションの起動時に実行されます。

ワンポイント：<application>要素の子要素として記述される<script>要素をデフォルトスクリプトと呼びます。

ステート変数の概念

ステート変数の値の変化は「アプリケーションの状態が変わった」、「アプリケーションの状態が遷移した」ことを意味します。例えば、紙芝居のような場面遷移を想像してください。

場面 A→場面 B

と遷移する場合、UJML ではそれぞれの場面に対してステート変数の値を対応させることができます。例えば次のようになります。

プログラムの場面	ステート変数 sScene の値
場面 A	1
場面 B	2

プログラム中で sScene の値に 1 を設定すると場面 A が表示され、sScene の値に 2 を設定すると場面 B が表示されるといった状態の管理をステート変数で行うことができます。

sscene.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sScene" type="int"/>
        </state-variables>
        <script>
            sScene = 1;
        </script>
        <states>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <state var="sScene">
                <transition value="1">
                    <!-- 場面 A -->
                </transition>
                <transition value="2">
                    <!-- 場面 B -->
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

sscene.ujml 内では各場面 A,B の具体的な定義はしていませんが、コメント（<!-- -->）の位置に場面 A,場面 B で表示する内容を定義することで場面を管理することができます。

ステート変数の値の変化

ステート変数の値が変化すると、それまでの値に対応して表示されていたものは全て非表示になります。ステート変数の変化により次のような内部的な処理が発生します。（sScene を例としています。）

ステート変数 sScene の変化	内部的な処理
1 で初期化	場面 A の内容を表示する。
1 から 2 に変化	場面 A の内容を非表示にし、場面 B の内容を表示する。
2 から 0 に変化	場面 B の内容を非表示にする。

ワンポイント：ステート変数が変化すると、それまでの値に対応して表示されていたものは全て非表示になります。

ステート変数の扱い

これまで説明したように、アプリケーションの状態を管理できるステート変数は他のプログラミング言語にある変数と違う特別なもののような感じがしますが、スクリプト中では他の言語の変数と同じように扱うことができます。

数値型のステート変数なら四則演算ができますし、文字列型のステート変数なら文字列の連結、切り取りといったことが可能です。また、ステート変数の型は数値型（int）、文字列型（string）、ブーリアン型（boolean）で宣言することができます。

ただし、文字列型（string）のステート変数はあまり使用しません。単純な状態を管理す

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

るだけなら、ブーリアン型（boolean）または数値型（int）を使用しましょう。

ワンポイント：文字列型（string）のステート変数はあまり使用しません。ステート変数は単純な状態を管理するだけなら、ブーリアン型（boolean）または数値型（int）を使用しましょう。

ステート変数と<transition>要素

さて、先ほど解説した state.ujml では sHelloWorld を true に設定しました。では、true に設定せずに false に設定したらどうなるでしょうか？

statefalse.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sHelloWorld" type="boolean"/>
        </state-variables>
        <script>
            sHelloWorld = false;
        </script>
        <states>
            <state var="sHelloWorld">
                <transition value="true">
                    <display>
                        <label>
                            <text>Hello World!</text>
                        </label>
                    </display>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

statefalse.ujml を実行しても画面上に文字列は表示されません。では次のように書き換えるとどうなるでしょうか？

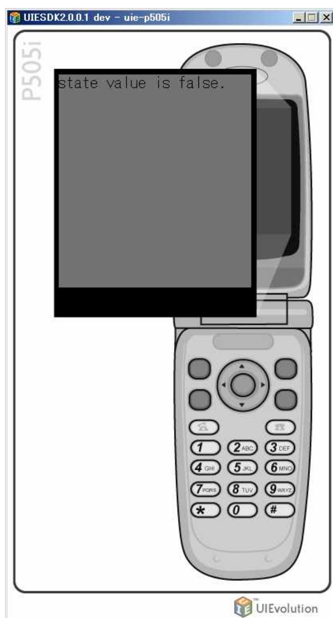
statefalse2.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <state-var name="sHelloWorld" type="boolean"/>
        </state-variables>
        <script>
            sHelloWorld = false;
        </script>
        <states>
            <state var="sHelloWorld">
                <transition value="true">
                    <display>
                        <label>
                            <text>Hello World!</text>
                        </label>
                    </display>
                </transition>
                <transition value="false">
                    <display>
                        <label>
                            <text>state value is false.</text>
                        </label>
                    </display>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

statefalse2.ujml を実行すると<transition>要素の value 属性値が false の部分で定義された文字列「`state value is false.`」が表示されます。



Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

先ほど使用した表で解説すると

場面（表示される文字列）	ステート変数 sHelloWorld の値
Hello World!	true
state value is false.	false

のようになります。ステート変数 sHelloWorld の値を切り替えることで、表示される文字列も切り替えることができました。

statefalse2.ujml ではブーリアン型のステート変数 sHelloWorld に false を設定したときに文字列を表示させましたが、通常、ブーリアン型のステート変数を使う場合には慣例として true を表示、false を非表示とするため、<transition>要素の value 属性の値が false の<display>要素は定義しません。ここでは、概念を理解するために敢えて false に<display>要素を定義しました。

ワンポイント：ブーリアン型のステート変数を使う場合には、<transition>要素の value 属性の値が false の<display>要素は定義しないようにしましょう。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

４．スクリプト

本章では、スクリプトについてより詳しく解説していきます。既にこれまで紹介したソースコード中にも出てきましたが、スクリプトは<script>要素内で普通のプログラミング言語のように自由に書くことができます。

例えば変数 mNum に 10+1 の結果を代入したければ、

```
mNum=10+1;
```

と書きます。

スクリプトの構文は JavaScript に良く似ていますが、次の点で特に異なる部分がありますので注意して読み進めましょう。

- 変数の宣言
- 比較演算子
- 定数

簡単なスクリプト

まず、簡単なスクリプトを紹介します。これまで紹介したサンプルは画面に表示させるものでした。今回はデバッガーに文字列を出力してみましょう。

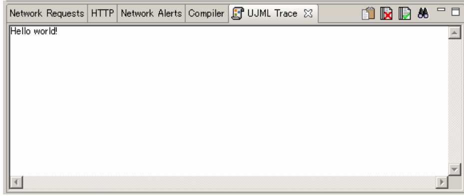
script.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <script>
            _trace("Hello world!");
        </script>
    </application>
</ujml>
```

このように、スクリプトは<script>要素の中に記述することで実行されます。デバッガーに文字列を出力するときには_trace()関数を使用します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

_trace()関数を使用すると、SDK の「UJML Trace」ビュー（下図）に文字を出力することができます。アプリケーションの実行状況を適宜出力すればデバッグしやすくなります。



変数

UJML では他の開発言語と同じように変数を使うことができます。変数は必ず宣言しなければなりません。宣言には<var>要素を使います。UJML には変数とステート変数という二種類の変数がありますが、ここでは普通の変数について解説します。

本ドキュメントにおいてステート変数は変数と区別できるように必ず「ステート変数」と表記します。また、単に「変数」と表記した場合には普通の変数を指しています。

変数の宣言

変数を宣言する場合には次のように記述します。

```
<var name=”変数名” type=”型” size=”配列インデックス数”/>
```

変数を配列にしない場合には size 属性を指定しません。int 型の変数 mVal を宣言する場合には次のように記述します。

```
<var name=”mVal” type=”int”/>
```

他のプログラム言語でも同じことが言えますが、変数にはわかりやすい意味のわかる単語の名前などつけるように心がけましょう。後から読み返したときに何をやっているのかさっぱりわからなくなってしまいますよ。

<application>要素の子要素として宣言した変数は全てグローバルな変数となります。UJML では関数の宣言内等でローカル変数を宣言することもできますが、<application>要素の子要素として宣言した変数のスコープはグローバルになることを覚えましょう。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

ワンポイント：<application>要素の子要素として宣言した変数はグローバル変数となります。

　また、グローバルな変数の名前の先頭には小文字のエム”m”を慣例的につけ、二文字目を大文字としています。これは、グローバルな変数とローカル変数（関数内、イベント内等）をすぐに区別するための知恵ですが、これに習うと良いでしょう。

ワンポイント：変数の名前はローカル変数と区別するために先頭に小文字のエム”m”をつけましょう。

型

　変数には次の型を使用することができます。

型	意味	初期値
boolean	true/false	false
string	文字列	空文字列
int	数値（整数）	0
リファレンス型	オブジェクトへの参照	null
インターフェイス型	インターフェイス	null

最後の二つのリファレンス型とインターフェイス型については本ドキュメントでは触れません。より高度なプログラミング及び設計をする際に使用します。

　表の通り、それぞれの変数は初期化しない場合初期値を持ちますが、念のため初期化することを推奨します。

ワンポイント：変数は初期値を持ちますが、初期化しましょう。

ステート変数

　既に基礎的な解説をしましたが、UJML ではステート変数という特殊な変数を使うことができます。UJML を特徴づけるこのステート変数はアプリケーションの状態を管理するために使用します。スクリプト中でステート変数は変数と同じように使用することができますが、ステート変数の値が変化するたびに変化した値に対応する処理がおこなわれます。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

ステート変数の宣言

　ステート変数を宣言する場合には次のように記述します。

```
<state-var name=”ステート変数名” type=”型” size=”配列インデックス数”/>
```

ステート変数を配列にしない場合には size 属性を指定しません。int 型のステート変数 sVal を宣言する場合には次のように記述します。

```
<state-var name=”sVal” type=”int”/>
```

ステート変数にもやはり変数と同様にわかりやすい名前をつけることを心がけましょう。

　また、ステート変数の名前の先頭には小文字のエス”s”を慣例的につけ、二文字目を大文字としています。これは、変数とステート変数をすぐに区別するための知恵ですが、これに習うと良いでしょう。

ワンポイント：ステート変数の名前は変数と区別するために先頭に小文字のエス”s”をつけましょう。

宣言されたステート変数は変数と同様にそのスコープはグローバルです。ただ、ステート変数は関数内等のローカルスコープでは宣言できませんので、名前でグローバル、ローカルを区別する必要は無いでしょう。

型

　ステート変数には次の型を使用することができます。

型	意味	初期値
boolean	true/false	false
string	文字列	空文字列
int	数値（整数）	0

　表の通り、それぞれの変数は初期化しない場合初期値を持ちます。

　３章でステート変数が変化すると対応する<transition>要素に定義されたヴィジュアル要素が表示され、スクリプトが実行されると解説しましたが、ステート変数が初期値の状態では該当する表示・実行処理は行われません。ステート変数をスクリプトで明示的に初

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

期化したときに初めて、対応する<transition>要素に定義されたヴィジュアル要素が表示され、スクリプトが実行されます。

ワンポイント：ステート変数は初期値を持ちますが、明示的に初期化されたときに初めて対応する<transition>要素の内容が実行されます。

また、既に解説しましたが、ステート変数に宣言できる型のうち、文字列型（string）のステート変数はあまり使用しません。単純な状態を管理するだけなら、ブーリアン型（boolean）または数値型（int）を使用しましょう。

ワンポイント：文字列型（string）のステート変数はあまり使用しません。ステート変数は単純な状態を管理するだけなら、ブーリアン型（boolean）または数値型（int）を使用しましょう。

変数及びステート変数の配列

UJML では他の開発言語と同じように配列を使うことができます。変数の場合には<var>要素に、ステート変数の場合には<state-var>要素に size 属性値を指定すれば配列のインデックス数を指定できます。

例えば、インデックス数 10 の int 型の配列 mArray を宣言する場合には、

```
<var name=”mArray” type=”int” size=”10”/>
```

のように記述します。配列の変数をプログラム中で使用する場合には次のように大括弧内にインデックス番号を指定します。

```
mArray[0] = 1;
```

静的な配列

静的な配列はあらかじめ配列のインデックス数が決まっている場合に使用し、配列のインデックス数をあらかじめ宣言します。次の例では int 型の配列 mArray をインデックス数 10 で宣言しています。

```
<var name=”mArray” type=”int” size=”10”/>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

動的な配列

動的な配列はあらかじめインデックス数が決められない場合に使用します。変数の宣言時に配列のインデックス数を 0 で宣言します。スクリプト中で関数_resizeArray()を使い配列のインデックス数を変化させることができます。ただし、ステート変数は動的な配列にすることはできません。必ず静的な配列としてください。

```
<var name="mArray" type="int" size="0"/>
```

```
<script>
    _resizeArray(mArray,10);
</script>
```

ワンポイント：ステート変数は動的な配列にすることはできません。

動的な配列は UJML2.0 から使用できるようになりました。それより前のバージョンでは使用できません。

コメント

プログラムの中にはコメントを書くことができます。UJML ではコメントを書くためにいくつかの方法が提供されています。コメントはプログラム中に書いても直接プログラムとしては扱われません。

プログラムのコメントとして、

- プログラムの目的
- プログラムの処理内容の説明

を日本語の文章で書くことができます。プログラムを読むのは書いた自分だけとは限りません。誰が読んでもわかりやすいように、コメントを残しましょう。

複数行コメント

<!-- -->

XML のコメントを使用し複数行にわたるコメントを挿入できます。上記タグに囲まれた全ての行はコメントとなります。このタグはソースコード中のいかなる場所にも挿入可能です。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
   <application>
      <display>
         <label><!-- 文字列を表示する -->
            <text>Hello World!</text>
         </label>
      </display>
   </application>
</ujml>
```

<script>内での 1 行コメント

// コメント

<script>要素中で他の言語と同じように 1 行コメントを記述できます。//を書いた箇所から行の終わりまでがコメントとなります。

```
      <script>
         // sHelloWorld = true に
         // 対応する<transition>要素内定義された
         // 文字列表示する。
         sHelloWorld = true;
      </script>
```

<script>内での複数行コメント

/* コメント */

<script>要素中で他の言語と同じように複数行コメントを記述できます。また、このコメントを使うことで行の一部をコメントにすることも可能です。

```
      <script>
         /*
            sHelloWorld = true に
            対応する<transition>要素内定義された
            文字列表示する。
         */
         sHelloWorld = true;
      </script>
```

コメントタグ

<comment>コメント</comment>

次の要素に対して<comment>要素を子要素として記述することができます。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

要素
interface
component
state-machine
var
function
template
state
transition

```
<functions>
    <function name="sum" type="int">
        <comment>引数の合計値を計算します</comment>
        <parameters>
            <var name="a" type="int"/>
            <var name="b" type="int"/>
        </parameters>
        <variables>
            <var name="ret" type="int"/>
        </variables>
        <script>
            ret = a + b;
        </script>
        <return><eval>ret</eval></return>
    </function>
</functions>
```

<comment>要素を使用しコメントを記述すると、コメント文は XML 文書として取り扱うことができるデータになります。この特性を活かし、例えばドキュメント生成の自動化を容易にすることができます。

定数　ENTITY

UJML では<ENTITY>要素を使うことで定数を使用することができます。<ENTITY>はソースコードの 2 行目にある DOCTYPE 宣言内で行います。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd" [

    <!-- 表示文字関係の定数 -->
    <!ENTITY START_MESSAGE "START" >
]>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

定数を使用する場合には次のようにします。まず、要素値として使う場合には次のように記述します。

```
<label>
   <text>&START_MESSAGE;</text>
</label>
```

ENTITY で宣言した”START_MESSAGE”に対して、先頭に&（アンド）、末尾に;（セミコロン）を付けて使います。

　スクリプト中で使う場合には、

```
<script>
   mValStr = “&START_MESSAGE;”;
</script>
```

のようにします。この例では mValStr は string 型の変数です。string 型の変数の値として定数を使う場合、&START_MESSAGE;自体をダブルクォーテーションで囲むことに注意しましょう。

ワンポイント：文字列定数を使う時には、定数自体をダブルクォーテーションで囲みましょう。

　int 型として使う場合には次のように書きます。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC “-//UIEVOLUTION//DTD UJML 2.1//EN”
“http://www.uievolution.com/dtd/ujml-2.1.dtd” [

   <!ENTITY MSEC_TIMER "10" >
]>
```

```
<script>
   mValInt = &MSEC_TIMER;;
</script>
```

ここで、セミコロンが二重になっていますが、忘れやすいので気をつけましょう。

　定数は UJML をコンパイルする際に、定数として指定した箇所の文字列が置換されることで実現されます。そのため、文字列を使う場合には“&START_MESSAGE;”のように定数をダブルクォーテーションで囲む必要があります。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

また、通常のソースコードは<ujml>要素内に記述しますが、定数の宣言は<ujml>要素の外に記述します。

ワンポイント：スクリプト中で定数が行の最後にあるとセミコロンを付け忘れやすいです。定数が行の最後にある場合にはセミコロンが二つ重なります。注意しましょう。

メモ：定数はコンパイル時に文字列が置換されることで実現されます。そのため、定数の宣言箇所にスクリプトを直接記述してもコンパイル可能です。ソースコードが読みにくくなるので推奨しません。

予約語

UJML ではいくつかの組み込み関数が定義されているため、変数や関数の名前として使えないものが存在します。組み込み関数は必ず_（アンダースコア）が先頭についているので、アンダースコアで始まる名前は使わないようにしましょう。

変数の要素値への使用

UJML では変数を要素値に使用することができます。変数を要素値に使用することで、例えば helloworld.ujml で表示した文字列をダイナミックに変更することができます。

それでは、最初に使用した「Hello World!」を表示するサンプル helloworld.ujml を応用し変数を使用して「Hello World!」を表示してみましょう。helloworld.ujml では<text>要素に表示したい文字列を直接指定していましたが、変数を使用する方法は次のようになります。

helloworldvar.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <variables>
            <var name="mHelloWorld" type="string"/>
        </variables>
        <script>
            mHelloWorld = "Hello World!!";
        </script>
        <display>
            <label>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <text><eval>mHelloWorld</eval></text>
        </label>
    </display>
</application>
</ujml>
```

まず<var>要素で変数 mHelloWorld を宣言します。さらに、<text>要素に<eval>要素を使い、要素値に変数 mHelloWorld を指定することで、変数に代入された文字列を表示することができます。<eval>要素内には変数名だけでなく、式（例えば 3＋2）を記述することもできます。

ワンポイント：要素値に変数や式を使う場合には<eval>要素を使います。さらに、<eval>要素内では変数名だけでなく式を指定することもできます。

この例のように、要素の値に変数の値を使用する場合には<eval>要素を使いますが、UJML では属性値に変数や式を適用する方法はありません。例えば、

```
<state name="sHelloWorld">
   <transition value="3 + mNum">
```

のようには記述することはできません。

ワンポイント：UJML では属性値に変数や式を指定できません。

条件判断　if

UJML では他の言語と同じように条件判断として if を使用できます。

```
if(i==1){
 _trace("i は 1 です。");
}
```

この例は変数 i の値が 1 のときにデバッガー上に出力されます。else 文を使うことで、次のように複雑な条件分岐をさせることも可能です。

```
if(条件１){

}else if(条件２){

}else if(条件３){
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
}else{

}
```

比較演算子

UJML では比較演算子として次のものが提供されています。

意味	関数	定数
> 大なり	_gt()	&_GT;
< 小なり	_lt()	&_LT;
>= 大なりイコール	_gte()	&_GTE;
<= 小なりイコール	_lte()	&_LTE;
== 等価	_eq()	なし ==をそのまま使う
!= 非等価	_ne()	なし !=をそのまま使う

比較演算子は表のように関数、定数の二種類提供されています。XML 要素のタグに使用される<>との衝突を避けるため、大なり">"、小なり"<"は使用できません。関数、定数のどちらを使っても全く同じ結果が得られます。

実際には、比較演算子として大なり">"は使用できます。ただし、対になる小なり"<"は使用できませんので使用しないことを推奨します。

例えば、「変数 i が 10 未満である」を評価する場合には、関数を使い、

```
if(_lt(i,10)){
 _trace(“i は 10 未満です。”);
}
```

とも書けますし、定数を使い、

```
if(i &_LT; 10)){
 _trace(“i は 10 未満です。”);
}
```

と書くこともできます。

仮に、小なり"<"使用すると次のようになりますが、

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

compare.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <variables>
            <var name="i" type="int"/>
        </variables>
        <script>
            i = 1;
            if(i <= 10){
                _trace(_strcat("i=",i));
            }
        </script>
    </application>
</ujml>
```

次のようなコンパイルエラーが発生します。

FATAL ERROR 'file:///C:/compare.ujml' at line 11 (org.apache.crimson.parser/P-079 20)

ワンポイント：比較演算子として大なり">"、小なり"<"は使わないようにしましょう。

繰り返し　while,for

UJML では他の言語と同じように繰り返しの処理を記述することができます。使用できる構文は while と for です。但し、他のプログラム言語にある繰り返し処理を抜けるための break 文を使用することはできません。

以下のサンプルコードは、繰り返し処理の内容を_trace()関数を使用してデバッガー上に出力しています。

まず、while を使う場合は次のように書きます。

while.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <variables>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <var name="i" type="int"/>
        </variables>
        <script>
            i=0;
            while(_lt(i,10)){
                _trace(_strcat("i=",i));
                i++;
            }
        </script>
    </application>
</ujml>
```

break を使えないので、繰り返し処理から抜ける条件は必ず while([条件])の条件に書かなければなりません。ここで条件に指定されている_lt(i,10)は i が 10 未満という条件です。

スクリプト中にある_strcat()関数は文字列を連結する関数です。

for を使う場合は次のように書きます。

for.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <variables>
            <var name="i" type="int"/>
        </variables>
        <script>
            for(i=0; _lt(i,10); i++){
                _trace(_strcat("i=",i));
            }
        </script>
    </application>
</ujml>
```

ワンポイント：繰り返し文では break を使うことはできません。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

５．ステート変数：応用編

本章ではステート変数のより高度な使い方について解説していきます。これまでにステート変数の簡単な使い方は解説しました。本章ではステート変数と<delay>要素、<script>要素を組み合わせる方法を中心に解説していきます。

ステート変数と delay と script

スクリプト中でステート変数に値を設定すると、対応する<transition>要素に定義されたヴィジュアル要素が表示され、スクリプトが実行されることは既に解説しました。その<transition>要素には子要素として次の要素を記述できます。

要素	目的	規定値
display	ヴィジュアル要素の表示	なし
play	オーディオ等の実行	なし
resources	リソース処理の実行	なし
delay	指定ミリ秒後にスクリプトを実行	なし
variables	スクリプト内で使用する変数の宣言	なし
execute	実行すべき XML スクリプト	なし
script	実行すべきスクリプト	なし

ここから、この中にある<delay>要素と<script>要素の使い方を解説していきます。

次のサンプル timer.ujml では一定間隔毎に処理を繰り返すことで、10 秒数えるタイマーを実現しています。

timer.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sTimer" type="boolean"/>
        </state-variables>
        <variables>
            <var name="mTime" type="int"/>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
        </variables>
        <script>
            mTime = 10;
            sTimer = true;
        </script>
        <states>
            <state var="sTimer">
                <transition value="true">
                    <display>
                        <label>
                            <text><eval>mTime</eval></text>
                        </label>
                    </display>
                    <delay>1000</delay>
                    <script>
                        mTime--;

                        if(mTime >= 0){
                            _clear_state(sTimer);
                            sTimer = true;
                        }
                    </script>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

<delay>要素に値 1000 が指定されています。これは、ステート変数 sTimer が true に設定された 1000 ミリ秒後に<script>要素内に記述されたスクリプトを実行するという意味です。

ワンポイント：実は<delay>要素に指定した時間が経過した後に<script>要素内のスクリプトが実行されることは保証されません。デバイスが他の処理をしている場合には指定時間より遅れて実行されることもあります。実際には、<delay>要素に指定した値は「○○ミリ秒後に実行キューにいれ、順番がきたら実行する」という意味になります。

さらに<script>要素内で、関数_clear_state()にステート変数 sTimer を渡しています。これは、それまでステート変数に設定されていた値 true に対応する<transition>要素内のヴィジュアル要素を非表示にすることを意味しています。ステート変数の値自体は変化しません。さらにその後すぐにステート変数 sTimer を true に設定することで、true に対応する<transition>要素内に定義されたヴィジュアル要素の表示とスクリプトの実行の繰り返しを実現しています。

このようにすることで、1000 ミリ秒毎にステート変数 sTimer が true に設定しなおされ、画面に mTime の値が表示されます。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

timer.ujml では_clear_state()関数で sTimer をクリアした後、明示的に sTimer に true を設定していますが、同様の処理を次のように書くこともできます。

```
_clear_state(sTimer);
sTimer = sTimer;
```

このように記述すると、sTimer の値を知らなくとも再度<transition>要素内に定義されたヴィジュアル要素を表示し、スクリプトを実行することができます。

また、変数 mTime を<text>要素の値として使用するために<eval>要素を使用しています。一度説明していますが、この<eval>要素の使い方は必ず覚えましょう。

<transition>要素

先ほど表で示した通り、<transition>要素の子要素には次のものがあります。

要素	目的	規定値
display	ヴィジュアル要素の表示	なし
play	オーディオ等の実行	なし
resources	リソース処理の実行	なし
delay	指定ミリ秒後にスクリプトを実行	なし
variables	スクリプト内で使用する変数の宣言	なし
execute	実行すべき XML スクリプト	なし
script	実行すべきスクリプト	なし

<transition>要素に対応する値がステート変数に設定されたときのアプリケーションの振る舞いはこの表にある目的の通りです。

本ドキュメントでは<play>要素の解説はしておりませんが、実際には<play>要素に定義された WAVE ファイル等が再生されます。

ステート変数が対応する値に設定されたときと、それ以外の値に変更されるか_clear_state()関数により無効になったときにどのようになるのかを次の表にまとめています。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

要素	対応する値に設定されたとき	値が変更されたとき または_clear_state()
display	ヴィジュアル要素が表示される。	ヴィジュアル要素が非表示になる。
play	オーディオ（WAVE ファイル）等が再生される。	再生が停止される。
resources	リソースをロードしメモリ中に展開される。	メモリから削除される。
delay	指定ミリ秒後にスクリプトを実行する。	なし
variables	スクリプト内で使用する変数の宣言	なし
execute	XML スクリプトが実行される。	なし
script	スクリプトが実行される。	なし

<display>要素、<play>要素、<resources>要素はステート変数が<transition>要素に対応する値に設定されると、それぞれ、ヴィジュアル要素の表示、オーディオの再生、リソースのロードが行われ、その状態がステート変数の値が変更されるまで維持されます。

厳密に言うと、オーディオに関しては最後まで再生されるとそこで再生は停止します。繰り返されるわけではありません。

<execute>要素、<script>要素はステート変数が<transition>要素に対応する値に設定されると、スクリプトの内容が一度だけ実行されます。先の要素と違い、何か状態が維持されるわけではありません。<variables>要素で宣言したローカル変数の値は全て実行の度に初期化されます。

ステート変数とヴィジュアル要素の描画

<transition>要素内にヴィジュアル要素を定義でき、対応するステート変数の値を設定することでヴィジュアル要素を表示できることは説明してきました。これまで紹介した中では、例えば、

```
<script>
   sHelloWorld = true;
</script>
```

のようにスクリプト中でステート変数に値を設定することで、ヴィジュアル要素を表示しました。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

実はステート変数に値が設定された瞬間にヴィジュアル要素が画面に描画されるわけでは無いことを覚えておく必要があります。ヴィジュアル要素が画面に描画されるのはアプリケーションがスクリプトを実行していない状態だけです。

もし、先ほどのステート変数 sHelloWorld に true を設定するスクリプトの直後に、非常に処理に時間のかかる長いループを書いた場合には、その実行が終了するまで画面にヴィジュアル要素が描画されません。

UJML ではヴィジュアル要素の画面描画がこのようなタイミングで行われるので、描画をブロックしてしまうようなスクリプト（例：長いループ）は書いてはいけません。

ワンポイント：ヴィジュアル要素の描画をブロックしてしまうような処理に時間のかかるスクリプトは書いてはいけません。ヴィジュアル要素の描画はスクリプトが何も実行されていないときに行われます。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

６．関数

本章では関数について解説していきます。関数はスクリプト中の同じような処理をまとめて書くときに便利です。また、UJML では便利な組み込み関数が提供されています。組み込み関数を活用することで強力なアプリケーションを開発できるようになるでしょう。

関数の基礎

UJML では他の言語と同じように独自の関数を定義できます。関数の定義には<function>要素を使用します。例えば、二つの int 型の引数の合計値を返す関数は次のように書きます。

```
<functions>
    <function name="sum" type="int">
        <comment>引数の合計値を計算します</comment>
        <parameters>
            <var name="a" type="int"/>
            <var name="b" type="int"/>
        </parameters>
        <variables>
            <var name="ret" type="int"/>
        </variables>
        <script>
            ret = a + b;
        </script>
        <return><eval>ret</eval></return>
    </function>
</functions>
```

独自の関数を書くときには<function>要素の属性と子要素を指定します。<function>要素の属性として指定するものは次の通りです。

属性名	目的	規定値
name	関数の名前	なし
type	関数の戻り値の型	なし
access	関数のローカルスコープ外への共有を指定	なし
visibility	関数のローカルスコープ外での可視性を指定	なし

メモ：access 属性と visibility 属性は複数のパーティションを使用するより高度な UJML アプリケーション開発に

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

求められるもので、本ドキュメントの対象外となります。パーティションについては付録をご覧ください。

また、子要素として指定するものは次の通りです。

要素名	目的	規定値
parameters	関数の引数	なし
script	関数内の処理	なし
return	関数の戻り値	なし

このように関数に定義すべき内容は他の開発言語と同じです。例えば引数に a と b を受け取り、合計値を戻す関数は次のようになります。

function.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <variables>
            <var name="num" type="int"/>
        </variables>
        <functions>
            <function name="sum" type="int">
                <comment>引数の合計値を計算します</comment>
                <parameters>
                    <var name="a" type="int"/>
                    <var name="b" type="int"/>
                </parameters>
                <variables>
                    <var name="ret" type="int"/>
                </variables>
                <script>
                    ret = a + b;
                </script>
                <return><eval>ret</eval></return>
            </function>
        </functions>
        <script>
            num = sum(10,1);//関数の呼び出し
            _trace( num );
        </script>
    </application>
</ujml>
```

この関数を呼び出すときには、

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
num = sum(10,1);
```

のように記述します。

組み込み関数を使う

UJML には多くの組み込み関数が用意されています。ここでは本ドキュメントに登場する組み込み関数のみを紹介します。

UJML の組み込み関数は必ずアンダースコア"_"で始まります。そのため、独自の関数を定義する際には関数名の先頭はアンダースコア以外の文字にしましょう。

_strcat()

_strcat()関数は全ての引数を連結した文字列を返します。

```
string = _strcat(string 文字列 1,string 文字列 2,string 文字列 3,,,);
```

引数は、リテラル文字列、変数、関数の戻り値等を指定することができます。連結させたい文字列を全て_strcat()関数の引数に渡すことができます。3 個の文字列を連結したい場合には 3 個の引数を指定します。また、boolean 型、int 型の値を引数に指定した場合には自動的に string 型に変換してから連結します。

_srand()

_srand()関数はシード値に基づいて int 型の乱数（正の整数）を返します。

```
int = _srand(int シード値);
```

_srand()関数は特定のシード値に対して必ず同じ結果を返します。アプリケーションの実行毎に異なる乱数が必要なときは、_msec()関数の返り値をシード値として渡すことを推奨します。また、連続的に乱数値が必要なときには、最初の_srand()関数の戻り値を次の_srand()関数のシード値にすることを推奨します。

_msec()

_msec()関数はデバイスの現在時刻を返します（単位：ミリ秒）。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
int = _msec();
```

_msec()関数はデバイスの現在時刻を返しますが、値の意味はデバイスによって異なります。基本的には UIEPlayer が起動してからの時間ですが、UTC 日付/時刻をサポートするデバイスでは、この関数が返す値は UIEPlayer が起動された日の午前 12 時（UTC 時間）からの経過時間です。

_getIntProperty()

_getIntProperty()関数は、数値のデバイス情報プロパティを int 値で返します。

```
int = _getIntProperty(int プロパティ ID);
```

_getIntProperty()関数は、数値のデバイス情報プロパティを取得するために使用します。本ドキュメントでもスクリーンの高さ、幅といったデバイス情報を取得する際に使用しています。

_text_width()

_text_width() 関数はある文字列を指定したフォントサイズ、フォントスタイルと字体を使って表示した場合の横幅を int 値で返します（単位：ピクセル）。

```
int = _text_width(int 文字列,int フォントサイズ,int フォントスタイル,int 字体);
```

_text_width() 関数は、指定した文字列の横幅を取得する関数です。この関数は<label>要素のサイズの計算に使用します。

_text_height()

_text_height() 関数は指定したフォントサイズ、フォントスタイルと字体を使用した場合の文字列の高さを int 値で返します（単位：ピクセル）。

```
int = _text_height(int フォントサイズ,int フォントスタイル,int 字体);
```

_text_height() 関数は、指定したフォントの高さを取得する関数です。この関数は<label>要素のサイズの計算に使用します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

_unload()

_unload() 関数は現在実行中のアプリケーションを停止し、直前に実行されていたヒストリースタック上のアプリケーションを読み込みます。

```
void _unload();
```

_unload()関数は現在実行中のアプリケーションを停止し、ヒストリースタックの一番上のアプリケーションバイトコードファイルを読み込み、実行します(アプリケーションの URL はヒストリースタックから降ろされます)。

ヒストリースタックについては付録で簡単な解説をしています。まずは_unload()関数は現在実行中のアプリケーションを停止すると覚えてください。

_clear_state()

_clear_state() 関数はステート変数の値を変更することなく、ステート変数の状態をクリアします。

```
void _clear_state(ステート変数名, [int index]);
```

_clear_state() 関数は引数に渡されたステート変数の現在の値に対応する<transition>要素内に定義されたヴィジュアル要素を非表示にします。例えば、ステート変数 sState が true で、対応する<transition>要素内で文字列を描画していたとき、

```
<state value="sState">
    <transition value="true">
        <display>
            <label><text>Hello World!</text></label>
…
```

_clear_state(sState)を実行すると、上記の<transition>要素内が無効化され<label>要素で表示されていた文字列が非表示になります。

ランダム値を作成する関数

ここから実際に記憶ゲームに必要な関数を作成していきます。ここで解説する

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

randRange()関数は_srand()関数を使用してランダム値を作成します。

srand.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <functions>
            <function name="randRange" type="int">
                <parameters>
                    <var name="seed" type="int"/>
                    <var name="min" type="int"/>
                    <var name="max" type="int"/>
                </parameters>
                <return>
                    <eval>
                        (_srand(seed) % (max - min + 1)) + min
                    </eval>
                </return>
            </function>
        </functions>
        <script>
            _trace(randRange(_msec(),1,100));
        </script>
    </application>
</ujml>
```

_srand()関数は

```
_srand(シード値)
```

のように使用します。ここでは、randRange()関数を宣言しています。この関数は指定された範囲の整数値を戻します。

　また_srand()関数のシード値としてデバイスの現在時刻を返す_msec()関数の返り値を使用しています。この方法は適当な乱数値を得る場合良く使う方法です。

UJML の_srand()関数は厳密な意味での乱数ではなく、シード値をもとに、ある計算を行い、その計算結果の値を返す関数です。そのため同じシード値を与えると必ず同じ値が返ります。

10 桁のランダム値を作成する関数

　先ほど作成した randRange()関数を使い、記憶ゲームで使う 10 桁のランダム値を生成す

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

る関数を作成してみましょう。

initNumber.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <functions>
            <function name="randRange" type="int">
                <comment>指定範囲の数字をランダムに作る関数</comment>
                <parameters>
                    <var name="seed" type="int"/>
                    <var name="min" type="int"/>
                    <var name="max" type="int"/>
                </parameters>
                <return>
                    <eval>
                        (_srand(seed) % (max - min + 1)) + min
                    </eval>
                </return>
            </function>
            <function name="initNumber" type="string">
                <comment>10 桁の数値（先頭ゼロ有り）の文字列を作成する関数</comment>
                <variables>
                    <var name="tmpSeed" type="int"/>
                    <var name="tmpNumber" type="string"/>
                    <var name="i" type="int"/>
                </variables>
                <script>
                    tmpSeed = _msec();
                    tmpNumber = "";
                    for(i=0; _lt(i,10); i++){
                        tmpNumber = _strcat(tmpNumber, randRange(tmpSeed,0,9));
                        tmpSeed = _srand(tmpSeed);
                    }
                </script>
                <return>
                    <eval>tmpNumber</eval>
                </return>
            </function>
        </functions>
        <script>
            _trace(initNumber());
        </script>
    </application>
</ujml>
```

この例では 0 から 9 の範囲の数値をできるだけランダムに出すために、_srand()関数で得られた値を、次の_srand()関数に渡すシード値として使用しています。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

連続的に乱数を取得したい場合にはこのように、前回の_srand()関数の結果を次の_srand()関数のシード値として使用する方法が推奨されています。そのとき最初の_srand()関数のシード値には_msec()関数の返り値を使うことが多いです。

ワンポイント：連続的に乱数を取得したい場合には、前回の_srand()関数の結果を次の_srand()関数のシード値として使用しましょう。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

７．ヴィジュアル要素の表示（<box>要素）

本章では<box>要素による長方形の描画方法の紹介を通し、ヴィジュアル要素の表示方法について解説していきます。UJML では図形描画に必要な、いくつかのヴィジュアル要素が提供されています。これらの要素を使うことで効率的なアプリケーション開発ができるでしょう。本章では<box>要素についてのみ解説します。

UJML では次のヴィジュアル要素が提供されています。

ヴィジュアル要素名	目的
box	長方形を描画する。
display-instance	コンポーネントのディスプレイ要素を描画する。
edit	1 行のテキスト入力を提供する。
fn	ファンクションキーの定義をする。
image	画像の全部、あるいは一部を描画する。
label	テキストを 1 行描画する。
line	始点と終点との間を結ぶ直線を描画する。
multi-edit	複数行のテキスト入力機能を提供する。
multi-label	複数行のテキストを描画する。
polygon	多角形を描画する。
polyline	互いに連結している複数の直線を描画する。
round-box	角の丸い長方形を描画する。
x-oval	楕円を描画する。

単純な長方形の描画

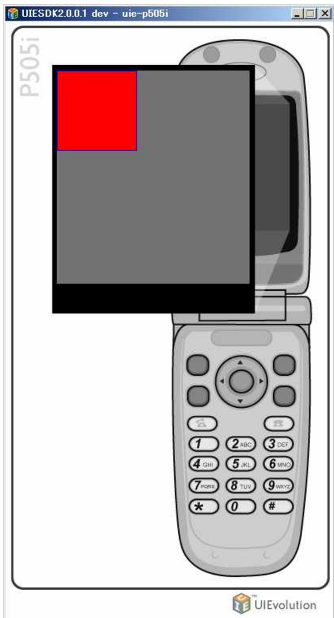
既にヴィジュアル要素の一つである<label>要素を使用した文字の表示を解説しましたが、ここから長方形の描画方法について解説していきます。まずは、簡単な長方形を表示します。

box.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
    <application>
        <display>
            <box>
                <width>100</width>
                <height>100</height>
                <fg>&_COLOR_BLUE;</fg>
                <bg>&_COLOR_RED;</bg>
            </box>
        </display>
    </application>
</ujml>
```



box.ujml では、長方形を、高さ、幅ともに 100 ピクセル、枠線を青色、背景を赤色で描画しています。

長方形の表示には<box>要素を使います。<box>要素の子要素に長方形の色、サイズ、位置等を指定することができます。子要素として指定できるものは次のものです。

要素名	目的	規定値
x	横方向の位置	0
y	縦方向の位置	0
width	幅	0
height	高さ	0
fg	幅 1 ピクセルの外枠の色	黒
bg	背景色	白

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

x-clip	子要素の表示範囲を box 内に限定するかどうか	false
event	イベント	なし
ヴィジュアル要素	子要素（長方形、ラベル等）	なし

<box>要素はサイズを指定しない場合は幅高さ共に規定値のゼロとなり、画面には何も表示されません。忘れずに指定しましょう。

ワンポイント：画面に表示させたい四角形は必ず width 属性と height 属性を指定しましょう。

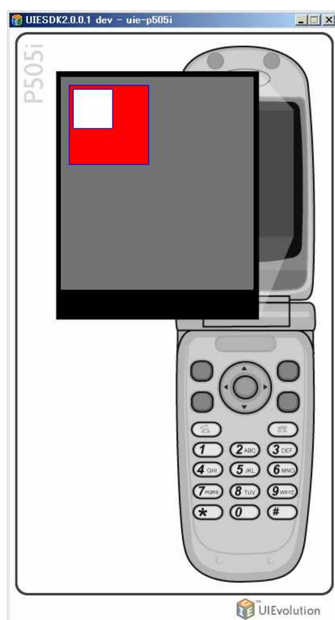
ヴィジュアル要素の入れ子構造

ヴィジュアル要素は入れ子構造とすることが可能です。例えば、大きな長方形の中に小さな長方形を描画するときは次のように記述します。

boxes.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <display>
            <box>
                <x>10</x>
                <y>10</y>
                <width>100</width>
                <height>100</height>
                <fg>&_COLOR_BLUE;</fg>
                <bg>&_COLOR_RED;</bg>
                <box>
                    <x>5</x>
                    <y>5</y>
                    <width>50</width>
                    <height>50</height>
                    <fg>&_COLOR_BLUE;</fg>
                    <bg>&_COLOR_WHITE;</bg>
                </box>
            </box>
        </display>
    </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止



大きな赤い長方形の中に小さい白い長方形を描画しています。赤い長方形は座標(X,Y)=(10,10)の位置に描画され、小さい白い長方形は<x>要素、<y>要素に 5 を指定していますが、実際には座標(X,Y)=(15,15)の位置に描画されています。これは、子要素の座標は親要素との相対座標となるからです。

　このように入れ子構造により相対座標を使えるため UJML では画面上のレイアウトをより簡単に行うことができます。

ワンポイント：ヴィジュアル要素が親要素にヴィジュアル要素を持つとき、座標は親要素からの相対座標になります。

Z-Order の話

　ヴィジュアル要素は UJML ソースコードに出現する順序に沿って、背面から前面に重ねられ描画されます。内部的に Z-ORDER という値で管理され、UJML ソース内の最初にある要素は Z-ORDER が低く、最後にある要素は Z-ORDER が自動的に高くなります。

　ヴィジュアル要素に明示的に Z-ORDER を指定することはできません。画面デザインを考える上では、ソースコード中に記述する要素の順序に配慮する必要があります。

ワンポイント：ヴィジュアル要素に Z-ORDER の値を明示的に指定することはできません。ソースコードに出現する順序に沿って、背面から前面に重ねられ描画されます。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

８．ヴィジュアル要素と変数

本章ではヴィジュアル要素に変数を使う方法を解説します。ヴィジュアル要素に変数を使うことで、四角形の位置や色、サイズを変化させる動きのあるアプリケーションを開発することができるでしょう。

<box>内に<label>を配置

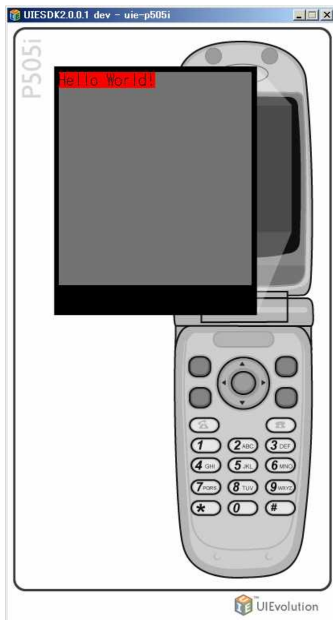
既に<label>要素を使い、簡単な文字列を表示する方法は解説しました。ここでは、<box>要素と<label>要素を組み合わせることで、文字列を装飾する方法を紹介します。

まず、次のソースコードを見てください。

labelbg.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC “-//UIEVOLUTION//DTD UJML 2.1//EN”
“http://www.uievolution.com/dtd/ujml-2.1.dtd”>
<ujml>
    <application>
        <display>
            <label>
                <text>Hello World!</text>
                <bg>&_COLOR_RED;</bg>
            </label>
        </display>
    </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止



labelbg.ujml を実行すると、<bg>要素に指定された赤色でラベルの背景が描画されます。ここで文字と背景の枠との間にマージン（隙間）を広く取りたい場合にはどうしたら良いでしょうか？

<label>要素にはマージンを指定する方法は提供されていません。このような場合<box>要素と<label>要素を組み合わせることで実現します。具体的に説明すると、背景となる<box>要素を描画し、その上に<box>要素の外枠からのマージンをとった位置に<label>要素を描画します。

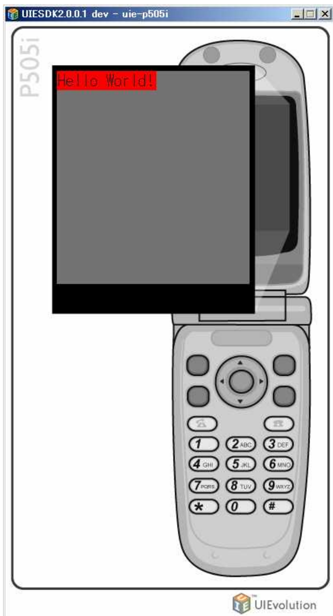
ソースコードを見てみましょう。

boxlabel.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <variables>
            <var name="mHeight" type="int"/>
            <var name="mWidth" type="int"/>
            <var name="mText" type="string"/>
        </variables>
        <script>
            mText = "Hello World!";
            // [+4] はマージンの分
            mWidth = _text_width(mText,
                                 &_FONT_SIZE_MEDIUM;,
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                                 &_FONT_STYLE_PLAIN;,
                                 &_FONT_FACE_SYSTEM;)
                                 + 4;
            mHeight = _text_height(&_FONT_SIZE_MEDIUM;,
                                   &_FONT_STYLE_PLAIN;,
                                   &_FONT_FACE_SYSTEM;)
                                   + 4;
        </script>
        <display>
            <box>
                <width><eval>mWidth</eval></width>
                <height><eval>mHeight</eval></height>
                <fg>&_COLOR_RED;</fg>
                <bg>&_COLOR_RED;</bg>
                <label>
                    <text><eval>mText</eval></text>
                    <x>2</x>
                    <y>2</y>
                </label>
            </box>
        </display>
    </application>
</ujml>
```



boxlabel.ujml では<box>要素の子要素に<label>要素があります。<label>要素の位置は親の<box>要素からの相対座標で指定されています。

ここで重要なのは、<box>要素の幅と高さの指定方法です。まず、デフォルトスクリプト内で表示する文字列の幅と高さを関数_text_width()、_text_height()を使用して取得しています。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

この値を元に、背景となる<box>要素の幅（<width>要素）と高さ（<height>要素）を指定しています。変数を要素値に指定する場合には必ず<eval>要素を使用します。

ワンポイント：変数を要素値に指定する場合には必ず<eval>要素を使用しましょう。

boxlabel.ujml ではマージンを 2 ピクセル取りました。

長方形のセンタリング

アプリケーションを作成する際、長方形や画像等を画面の中央に配置することが良くあります。ここでは長方形を画面中央に配置する方法を紹介します。

まずはソースコードをご覧ください。

boxcenter.ujml

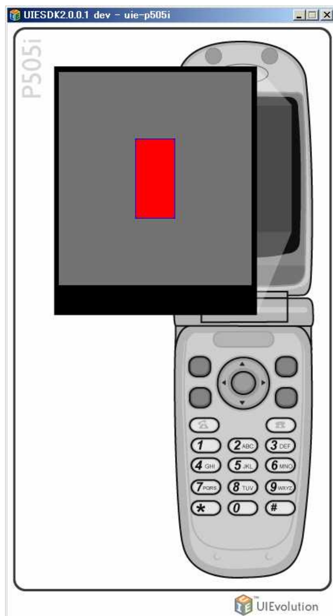
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <variables>
            <var name="mScrWidth" type="int"/>
            <var name="mScrHeight" type="int"/>
            <var name="mPosX" type="int"/>
            <var name="mPosY" type="int"/>
        </variables>
        <script>
            // スクリーンの幅、高さを取得する
            mScrWidth = _getIntProperty(&_PROPERTY_INT_SCREEN_WIDTH;);
            mScrHeight = _getIntProperty(&_PROPERTY_INT_SCREEN_HEIGHT;);

            // スクリーンのサイズを元に長方形を描画する
            // 位置を計算する。
            // 長方形のサイズは幅 50、高さ 100 とする。
            mPosX = (mScrWidth - 50) / 2;
            mPosY = (mScrHeight - 100) / 2;

        </script>
        <display>
            <box>
                <x><eval>mPosX</eval></x>
                <y><eval>mPosY</eval></y>
                <width>50</width>
                <height>100</height>
                <fg>&_COLOR_BLUE;</fg>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                <bg>&_COLOR_RED;</bg>
            </box>
        </display>
    </application>
</ujml>
```



画面の中央に配置するには画面のサイズ（幅、高さ）が必要です。画面のサイズを取得するには_getIntProperty()という関数を使用します。この関数に渡す引数により幅、高さを得ることができます。

```
mScrWidth = _getIntProperty(&_PROPERTY_INT_SCREEN_WIDTH;);
mScrHeight = _getIntProperty(&_PROPERTY_INT_SCREEN_HEIGHT;);
```

渡すべき引数は定数として用意されているものを使用します。

ここでは長方形を幅 50、高さ 100 としました。画面中央に描画するには長方形の X,Y の座標をそれぞれ次のように計算します。

```
X=([画面の幅] - [長方形の幅]) /  2
Y=([画面の高さ] - [長方形の高さ]) /  2
```

ソースコード中では、

```
mPosX = (mScrWidth - 50) / 2;
mPosY = (mScrHeight - 100) / 2;
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

となっています。ここで計算した値を<box>要素の座標に指定していますが、この際もちろん<eval>要素を使い、要素値に変数を指定します。

ワンポイント：変数を要素値に指定する場合には必ず<eval>要素を使用しましょう。

UJML における初期化の定石

UJML のデフォルトスクリプトはアプリケーションの初期化によく使用されます。アプリケーションの初期化では、

- 画面サイズ（高さ、幅）等の取得
- 文字列サイズ（高さ、幅）の取得

といった処理を行います。ここでは良く行われる初期化を解説します。

メモ：デフォルトスクリプトでは画面サイズだけでなく、各種デバイス情報も取得することが多いです。本ドキュメントは画面サイズ以外の情報は使用しないので解説しておりません。

画面サイズの取得

まず、画面サイズ（高さ、幅）の取得には_getIntProperty()関数を使用します。この関数は引数によって色々なデバイスの情報を取得することができます。画面サイズを取得する場合には引数に定数を渡し、

```
mScrWidth = _getIntProperty(&_PROPERTY_INT_SCREEN_WIDTH;);
mScrHeight = _getIntProperty(&_PROPERTY_INT_SCREEN_HEIGHT;);
```

とします。_getIntProperty()関数は引数により様々なデバイスの情報を取得することが可能です。また、同様の関数として_getStringProperty()関数というものもあります。_getStringProperty()関数は文字列で返るデバイス情報を取得するために使用します。

文字列サイズの取得

文字列サイズ（高さ、幅）の取得には_text_width()関数及び_text_height()関数を使用します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
W = _text_width(文字列, フォントサイズ, フォントスタイル, フォント種類)
H = _text_height(フォントサイズ, フォントスタイル, フォント種類)
```

フォントのサイズ、スタイル、種類は組み込みの定数を指定します。

フォントサイズで指定できる定数

定数	目的
&_FONT_SIZE_SMALL;	小
&_FONT_SIZE_MEDIUM;	中（標準の大きさ）
&_FONT_SIZE_LARGE;	大

フォントスタイルで指定できる定数

定数	目的
&_FONT_STYLE_PLAIN;	標準
&_FONT_STYLE_BOLD;	太字
&_FONT_STYLE_ITALIC;	イタリック（斜体）
&_FONT_STYLE_UNDERLINED;	下線付き
&_FONT_STYLE_OUTLINED;	アウトラインフォント（ほとんどのデバイスで未サポート）

フォント種類で指定できる定数

定数	目的
&_FONT_FACE_SYSTEM;	標準のシステムフォント

スタートボタンの作成

これまでの内容を参考にして画面上に表示するスタートボタンを作成してみましょう。スタートボタンは次のようなものとします。

スタートボタン：以下の参考図のように画面の中央に「START」という文字列が表示される長方形で、マージンを 4 ピクセル、枠線を 2 ピクセルとします。

参考図

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

START

これまで解説した内容を確認する意味でも、ソースコードを見る前に作ってみましょう。

startbutton.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd" [
    <!-- 表示文字関係の定数 -->
    <!ENTITY START_MESSAGE "START" >

    <!-- デザイン関係の定数 -->
    <!ENTITY SPACING "4">
    <!ENTITY BORDER_SIZE "2">
    <!ENTITY BORDER_COLOR "&_COLOR_BLUE;">
    <!ENTITY BACK_COLOR "&_COLOR_SILVER;">
    <!ENTITY FONT_COLOR "&_COLOR_BLACK;">

]>
<ujml>
    <application>
        <variables>
            <!-- 幅、高さ、位置 -->
            <var name="mScrWidth" type="int" />
            <var name="mScrHeight" type="int" />
            <var name="mPosX" type="int" />
            <var name="mPosY" type="int" />
            <var name="mWidth" type="int"/>
            <var name="mHeight" type="int"/>
            <!-- 解答欄に表示する文字 -->
            <var name="mBoxText" type="string"/>
        </variables>
        <script>
            <!-- スクリーンのサイズを調べる -->
            mScrWidth = _getIntProperty(&_PROPERTY_INT_SCREEN_WIDTH;);
            mScrHeight = _getIntProperty(&_PROPERTY_INT_SCREEN_HEIGHT;);

            <!-- 開始ボタンの全文字数分の幅 -->
            mWidth = (&SPACING; * 2) +
                _text_width("&START_MESSAGE;", &_FONT_SIZE_MEDIUM;,
                &_FONT_STYLE_BOLD;, &_FONT_FACE_SYSTEM;) +
                &BORDER_SIZE; * 2;

            <!-- 高さ：START ボタン及び解答欄 -->
            mHeight = (&SPACING; * 2) +
                _text_height(&_FONT_SIZE_MEDIUM;, &_FONT_STYLE_ITALIC;,
                &_FONT_FACE_SYSTEM;) +
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

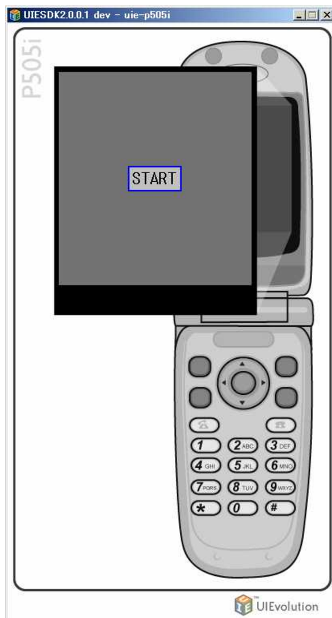
```
            &BORDER_SIZE; * 2;

        <!-- ラベルをセンタリングする X は都度計算 Y のみここで初期化-->
        mPosX = (mScrWidth - mWidth) / 2;
        mPosY = (mScrHeight - mHeight) / 2;

        <!-- 表示する文字の指定 -->
        mBoxText = "&START_MESSAGE;";

    </script>
    <display>
        <box><!-- 枠線を太く取るための四角形 -->
            <x><eval>mPosX</eval></x>
            <y><eval>mPosY</eval></y>
            <width><eval>mWidth</eval></width>
            <height><eval>mHeight</eval></height>
            <fg>&BORDER_COLOR;</fg>
            <bg>&BORDER_COLOR;</bg>
            <box><!-- マージンを取るための四角形 -->
                <x><eval>&BORDER_SIZE;</eval></x>
                <y><eval>&BORDER_SIZE;</eval></y>
                <width><eval>mWidth - &BORDER_SIZE; * 2</eval></width>
                <height><eval>mHeight - &BORDER_SIZE; * 2</eval></height>
                <fg>&BACK_COLOR;</fg>
                <bg>&BACK_COLOR;</bg>
                <label><!-- START という文字を表示する -->
                    <text><eval>mBoxText</eval></text>
                    <x>&SPACING;</x>
                    <y>&SPACING;</y>
                    <fg>&FONT_COLOR;</fg>
                    <style>&_FONT_STYLE_BOLD;</style>
                </label>
            </box>
        </box>
    </display>
  </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止



いかがでしょうか？全く同じソースコードになることは無いと思います。マージンを取った文字の表示及びセンタリングが正しくできましたか？startbutton.ujml は次の方針で作成しました。

- デフォルトの<script>要素内でスタートボタンの位置、サイズを初期化
- ENTITY 部分の値を変更することでデザインを変更可能
- スタートボタンのヴィジュアル要素は再利用できるように、プロパティは変数で指定する。

この例では、定数 ENTITY で指定した値を変更しても自動的に中央に描画できるようにしています。

また、ヴィジュアル要素にできるだけ変数を指定することで、アプリケーション内で他に流用できるようになります。例えば、startbutton.ujml では表示する文字の内容を「Try Again」とすれば再開ボタンにすることができます。

仮にスタートボタンを表示するだけなら「変数を指定する」必要はありませんが、今後の応用の意味も含めてここで紹介しました。

ワンポイント：ヴィジュアル要素に変数を指定することで、アプリケーション内で他に流用できるようになります。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

９．イベントの処理

本章では UJML のイベントについて解説していきます。「ユーザーのボタン入力により画面に表示するものを変化させる」といったインタラクティブなアプリケーションを開発するために必要なイベントの処理方法を解説していきます。

キー入力によるイベントの処理

まず、helloworld.ujml を改良し、ユーザーのキー入力によるイベントの処理を見てみましょう。

helloworldevent.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sHelloWorld" type="boolean"/>
        </state-variables>
        <script>
            sHelloWorld = true;
        </script>
        <states>
            <state var="sHelloWorld">
                <transition value="true">
                    <display>
                        <label>
                            <text>Hello World!</text>
                            <event name="onSelect">
                                <accelerators>
                                    <key>FIRE</key>
                                </accelerators>
                                <script>
                                    sHelloWorld = false;
                                </script>
                            </event>
                        </label>
                    </display>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

このサンプルでは、FIRE キーの入力により文字「HelloWorld」が非表示になります。<key>要素に指定された FIRE キーが入力されたとき<script>要素内のスクリプトが実行され、ステート変数 sHelloWorld を false に設定し、文字が非表示になります。

ここで<event>要素の書き方を簡単に解説します。<event>要素は

```
<event name="イベント名">
    <accelerators>
        <key>イベントに割り当てるキー</key>
        <key>イベントに割り当てるキー</key>
    </accelerators>
    <script>
        イベント発生時に処理されるスクリプト
    </script>
</event>
```

のように記述します。ユーザーの入力を受け付けるにはイベント名に onselect を指定します。さらにイベントに割り当てる入力キーを<key>要素に指定します。イベント発生時に実行されるスクリプトは<script>要素内に記述します。

もう一つ全く同じ動作をする例を見てみましょう。同じように FIRE キーの入力により文字が非表示になりますが、実装が異なります。

helloworldevent2.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sHelloWorld" type="boolean"/>
        </state-variables>
        <script>
            sHelloWorld = true;
        </script>
        <display>
            <box>
                <event name="onSelect">
                    <accelerators>
                        <key>FIRE</key>
                    </accelerators>
                    <script>
                        sHelloWorld = false;
                    </script>
                </event>
            </box>
        </display>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
        <states>
            <state var="sHelloWorld">
                <transition value="true">
                    <display>
                        <label>
                            <text>Hello World!</text>
                        </label>
                    </display>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

helloworldevent2.ujml では<label>要素には<event>要素を指定していません。別の<box>要素に指定しました。これでも同じように動作します。<box>要素の<width>要素、<height>要素に何も指定していませんので<box>要素は画面上に表示されません。画面上には表示されませんが、このようにイベント処理を行うことができます。

　このようになイベント処理のみ行う画面に表示されない<box>要素でユーザー入力イベントを処理する方法は良く使います。

ワンポイント：イベント処理のみを行う画面に表示されない<box>要素でユーザー入力イベントを処理することができます。

　ところで、

```
                    <accelerators>
                        <key>FIRE</key>
                    </accelerators>
```

の部分で気付いた方もいらっしゃると思いますが、<key>で指定する値を別のもの、例えば LEFT や RIGHT に変更することでイベントを割り当てるキーを指定することができます。

```
                    <accelerators>
                        <key>LEFT</key>
                    </accelerators>
```

　また、複数のキーを割り当てることも可能です。

```
                    <accelerators>
                        <key>FIRE</key>
                        <key>RIGHT</key>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <key>LEFT</key>
        </accelerators>
```

この場合は、FIRE,RIGHT,LEFT のどれか一つのボタン入力があったときにイベント処理されます。同時ではありません。同時入力をサポートする仕組はUJML の言語仕様標準では提供されていません。

<key>要素に指定できるキーは次の通りです。

キーの名前	詳細
0-9	数字キー
a-z	アルファベットの小文字キー
A-Z	アルファベットの大文字キー
UP, DOWN, LEFT, RIGHT	方向キー
FIRE	一部のデバイスでは、方向キー中央に配置されたキー。 その他のデバイスではリターンキー、エンターキー。等
GAME_A, GAME_B, GAME_C, GAME_D	ゲーム専用キー
F1, F2	ファンクションキー1 とファンクションキー2
POUND	シャープキー #
STAR	アスタリスクキー *
BACKSPACE	デバイスに依存
CLEAR	デバイスに依存

また、<event>要素は一つのヴィジュアル要素に対して一つしか記述できません。処理したいイベントが複数ある場合には、その数だけヴィジュアル要素を記述する必要があります。

ワンポイント：ヴィジュアル要素一つに<event>要素は一つしか記述できません。処理したいイベントの数だけヴィジュアル要素を記述しましょう。

ファンクションキーによるイベントの処理

UJML では<fn>要素を使いファンクションキーを定義することができます。ファンクションキーはアプリケーションのスクリーン外に表示されるボタンのことで、携帯電話では画面下部、左右に表示されるボタンのことを指します。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

ファンクションキーは<display>要素内で定義します。

buttoninput.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <display>
            <fn>
                <text>終了</text>
                <event name="onselect">
                    <accelerators>
                        <key>F1</key>
                    </accelerators>
                    <script>
                        _unload();
                    </script>
                </event>
            </fn>
        </display>
    </application>
</ujml>
```

<text>要素で機能ボタン上に表示する文字を指定します。<event>要素で関連付けるキーと実行するスクリプトを記述します。

ファンクションキーはフォントのサイズ、スタイルを指定することはできません。

ファンクションキーと関連付けるキーは F1,F2 がサポートされています。UJML2.1 では 2 個を越えるファンクションキーはサポートされていません。

数値ボタンによる入力イベントをチェックする話、配列使う

ここでは、0～9 の数字キーの入力を受け付ける方法を紹介します。

buttoninput.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd" [
    <!-- 動作関係の定数 -->
    <!ENTITY PAD_BUTTON_COUNT "10">
]>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
<ujml>
    <application>
        <state-variables>
            <state-var name="sButton" type="boolean" size="&PAD_BUTTON_COUNT;"/>
        </state-variables>
        <functions>
            <function name="initButton" type="void">
                <comment>入力ボタンを初期化する</comment>
                <parameters>
                    <var name="visible" type="boolean"/>
                </parameters>
                <variables>
                    <var name="i" type="int"/>
                </variables>
                <script>
                    for(i=0; _lt(i,&PAD_BUTTON_COUNT;); i++){
                        sButton[i] = visible;
                    }
                </script>
            </function>
        </functions>
        <script>
            initButton(true);
        </script>
        <states>
            <state var="sButton" index="*">
                <transition value="true">
                    <comment>入力を受け付ける box 要素を作成する</comment>
                    <display>
                        <box>
                            <event name="onSelect">
                                <accelerators>
                                    <key><eval>_state_index(0)</eval></key>
                                </accelerators>
                                <script>
                                    _trace(_state_index(0));
                                </script>
                            </event>
                        </box>
                    </display>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

まず、ステート変数 sButton を配列として宣言しています。インデックス数は 0〜9 のキーの数 10 としました。<state-var>要素の size 属性値にはファイルの先頭で定義した定数で指定しています。

```
        <state-variables>
            <state-var name="sButton" type="boolean" size="&PAD_BUTTON_COUNT;"/>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
        </state-variables>
```

ステート変数 sButton の<state>要素に注目してください。index 属性値がアスタリスク*で指定されています。

```
        <states>
            <state var="sButton" index="*">
                <transition value="true">
```

これは、ステート変数配列を使用する場合の重要なテクニックの一つで、この方法を使うとソースコードを簡潔にまとめることができます。では、何をまとめているのかというと、上記の方法でまとめたコードは下のコードと全く同じ意味を表します。

```
        <states>
            <state var="sButton" index="0">
                <transition value="true">
                    <comment>入力を受け付ける box 要素を作成する</comment>
                    <display>
                        <box>
                            <event name="onSelect">
                                <accelerators>
                                    <key><eval>0</eval></key>
                                </accelerators>
                                <script>
                                    _trace(0);
                                </script>
                            </event>
                        </box>
                    </display>
                </transition>
            </state>
            <state var="sButton" index="1">
                <transition value="true">
                    <comment>入力を受け付ける box 要素を作成する</comment>
                    <display>
                        <box>
                            <event name="onSelect">
                                <accelerators>
                                    <key><eval>1</eval></key>
                                </accelerators>
                                <script>
                                    _trace(1);
                                </script>
                            </event>
                        </box>
                    </display>
                </transition>
            </state>
            <state var="sButton" index="2">
            … 以下 2～9 まで続く
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
        </states>
```

つまり、本来 10 回書かなければならない<state>要素を 1 回にまとめています。こうすることで、ソースコードサイズを小さくでき、ソースコードが読みやすくなります。

<state>要素の index 属性値にアスタリスクを指定し、ステート変数配列のインデックス番号に対応する箇所に_state_index()関数を使うことで、このようにソースコードをまとめることができます。これは覚えておきたいテクニックの一つです。

ただしどちらの方法で書いても、コンパイル後のバイトコードのファイルサイズは変わりませんのでご注意ください。

ワンポイント：index=”*”を使いソースコードを簡潔にまとめても、コンパイル後のバイトコードのサイズは変わりません。

また、index=”*”の記述方法を使いソースコードを簡略化する場合には、<state>要素内に記述するスクリプトを必ず関数化しましょう。例えば、

```
        <states>
            <state var="sButton" index="*">
                <transition value="true">
                    <comment>入力を受け付ける box 要素を作成する</comment>
                    <display>
                        <box>
                            <event name="onSelect">
                                <accelerators>
                                    <key><eval>_state_index(0)</eval></key>
                                </accelerators>
                                <script>
                                    processInput(_state_index(0));
                                </script>
```

のようにし、processInput()関数内で処理を行うようにします。関数化しないと、スクリプトの処理内容が配列のインデックス数分展開され、コンパイル後のバイトコードが無駄に大きくなってしまいます。

ワンポイント：index=”*”とした<state>要素内のスクリプトは必ず関数化しましょう。

ステート変数の変化によるイベントの処理

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

既に簡単に紹介しましたが、<transition>要素に対し<script>要素を指定することで、ステート変数の変化によるイベントの処理を記述することができます。ユーザー入力等の外部的なイベントではなく、内部的にイベントを発生させるときにはステート変数を使います。

ここでは記憶ゲームの流れに沿ってゲームのフローをステート変数の値を変化させながら切り替えるサンプルを紹介します。ゲーム中は以下の 4 つの状態を繰り返します。

１．ゲーム開始（スタートボタンを表示）
２．問題の出題中
３．解答中
４．ゲーム終了（おめでとうを表示）

この状態の遷移を int 型のステート変数 sStatus を使って簡単に実現します。

statevariableevent.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd" []>
<ujml>
    <application>
        <state-variables>
            <state-var name="sStatus" type="int"/>
        </state-variables>
        <script>
            <!-- ゲーム開始 -->
            sStatus = 0;
        </script>
        <display>
            <box>
                <event name="onSelect">
                    <accelerators><key>FIRE</key></accelerators>
                    <script>
                        sStatus++;
                    </script>
                </event>
            </box>
        </display>
        <states>
            <state var="sStatus">
                <transition value="0">
                    <comment>START ボタン表示</comment>
                    <script>
                        _trace("START ボタンを表示しています。");
                    </script>
                </transition>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <transition value="1">
                <comment>数字を暗記する状態</comment>
                <script>
                    _trace("問題を出題中です。10 秒間で消える。");
                </script>
            </transition>
            <transition value="2">
                <comment>解答する状態</comment>
                <script>
                    _trace("解答中です。");
                </script>
            </transition>
            <transition value="3">
                <comment>おめでとう状態</comment>
                <script>
                    _trace("正解して終わりました。");
                </script>
            </transition>
        </state>
    </states>
  </application>
</ujml>
```

FIRE キーを入力すると状態（sStatus）が変化し、デバッガーに_trace()関数で指定された文字列が出力されます。

これまでのステート変数を使ったサンプルでは、ヴィジュアル要素の表示/非表示を切り替えることが中心でしたが、このように、ステート変数により内部的にアプリケーションの状態を遷移させることができます。

ゲームの完成

ここまでで、記憶ゲームを作成するために必要な UJML の知識を学ぶことができました。それでは、記憶ゲームを作成していきましょう。

ゲームの仕様
ゲーム中は以下の 4 つの状態を繰り返します。 １．ゲーム開始（スタートボタンを表示、FIRE ボタンで出題中へ） ２．出題中（10 秒間で自動的に解答中へ） ３．解答中（キー入力をチェックし全桁正解したらゲーム終了へ） ４．ゲーム終了（おめでとうを表示、1 秒でゲーム開始へ） 記憶する文字は 10 桁の数字（0～9）とする。但し、先頭がゼロもありうる。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

先程、ステート変数でアプリケーションの状態を管理するサンプルを紹介しましたが、記憶ゲームではステート変数 sStatus（int 型）を使い、次のようにアプリケーションの状態を定義しています。

sStatus の値	状態
0	ゲーム開始
1	出題中
2	解答中
3	ゲーム終了

記憶ゲームでは 6 個のステート変数が宣言され、それぞれ次の役割をしています。

ステート変数名	役割
sBackGround	背景の描画
sStatus	ゲームの状態の管理
sTimer	タイマー（出題中の 10 秒をカウント）
sBox	スタートボタン表示、解答欄の表示
sButton（配列 インデックス数 10）	解答中の入力を受け付ける
sEnding	終了時のメッセージの表示

ステート変数 sStatus の値と処理内容を表にまとめました。

sStatus の値	状態
初期状態	各種初期化（画面サイズ等） sBackGround を true に設定し背景を描画 sStatus を 0 に設定しゲーム開始
0	スタートボタンを表示 FIRE ボタンの入力により sStatus を 1 に設定
1	問題の 10 桁の数字を作成 sBox を true に設定し問題を表示 sTimer を true に設定し 10 秒のカウント開始 10 秒たったら sStatus を 2 に設定
2	解答欄を初期化 sBox を true に設定し解答欄を表示

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

	sButton を true に設定しボタン入力可能に ボタン入力毎に正誤チェック、全桁解答したら sStatus を 3 に設定
3	sEnding を true に設定しメッセージを表示 1 秒後に自動的に sStatus を 0 に設定

記憶ゲームでは、ゲーム全体の流れを管理するステート変数 sStatus と、ゲームに必要な表示要素やイベント処理要素のためのステート変数 sBackGround, sTimer, sBox, sButton, sEnding を宣言しています。ステート変数 sStatus の値の変化により、その他のステート変数の値を設定しています。

main.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd" [

    <!-- 表示文字関係の定数 -->
    <!ENTITY START_MESSAGE "START" >

    <!-- デザイン関係の定数 -->
    <!ENTITY SPACING "4">
    <!ENTITY BORDER_SIZE "2">
    <!ENTITY BORDER_COLOR "&_COLOR_BLUE;">
    <!ENTITY BACK_COLOR "&_COLOR_SILVER;">
    <!ENTITY FONT_COLOR "&_COLOR_BLACK;">

    <!-- 動作関係の定数 -->
    <!ENTITY SEC_LIMIT "10"><!-- 覚える秒数 -->
    <!ENTITY PAD_BUTTON_COUNT "10">

    <!ENTITY CHAR_SIZE "10" ><!-- 問題の桁数 -->

]>
<ujml>
    <application>
        <state-variables>
            <state-var name="sBackGround" type="boolean"/>
            <state-var name="sStatus" type="int"/>

            <state-var name="sTimer" type="boolean"/>
            <state-var name="sBox" type="boolean"/>
            <state-var name="sButton" type="boolean" size="&PAD_BUTTON_COUNT;"/>
            <state-var name="sEnding" type="boolean"/>

        </state-variables>
        <variables>
            <var name="mScrWidth" type="int" />
            <var name="mScrHeight" type="int" />

            <var name="mPosX" type="int" />
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <var name="mPosY" type="int" />

            <var name="mStartWidth" type="int"/>
            <var name="mHeight" type="int"/>

            <var name="mTimer" type="int"/>
            <var name="mSeed" type="int"/>
            <var name="mIndex" type="int"/><!-- 解答の位置 -->

            <var name="mBoxWidth" type="int"/><!-- 解答欄等の幅 -->
            <var name="mBoxText" type="string"/><!-- 解答欄に表示する文字 -->

            <var name="mAnswerWidth" type="int"/>
            <var name="mPosXAnswer" type="int"/>
            <!-- 正解の文字 -->
            <var name="mTrueAnswerChar" type="string" size="&CHAR_SIZE;"/>

            <var name="mBackColor" type="int"/>
            <var name="mBadColor" type="int"/>
            <var name="mNormalColor" type="int"/>

            <var name="mImageReady" type="boolean"/>
        </variables>
        <functions>
            <function name="randRange" type="int">
                <comment>指定範囲の数字をランダムに作る関数</comment>
                <parameters>
                    <var name="seed" type="int"/>
                    <var name="min" type="int"/>
                    <var name="max" type="int"/>
                </parameters>
                <return>
                    <eval>
                        (_srand(seed) % (max - min + 1)) + min
                    </eval>
                </return>
            </function>
            <function name="initNumber" type="string">
                <comment>問題の数値を作成する関数</comment>
                <variables>
                    <var name="tmpSeed" type="int"/>
                    <var name="tmpNumber" type="string"/>
                    <var name="i" type="int"/>
                </variables>
                <script>
                    tmpSeed = _msec();
                    tmpNumber = "";
                    for(i=0; _lt(i,&CHAR_SIZE;); i++){
                        tmpNumber = _strcat(tmpNumber, randRange(tmpSeed,0,9));
                        mTrueAnswerChar[i] = randRange(tmpSeed,0,9);
                        tmpSeed = _srand(tmpSeed);
                    }
                </script>
                <return>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <eval>tmpNumber</eval>
        </return>
    </function>
    <function name="initButton" type="void">
        <comment>0～9 の入力ボタンを初期化する</comment>
        <parameters>
            <var name="visible" type="boolean"/>
        </parameters>
        <variables>
            <var name="i" type="int"/>
        </variables>
        <script>
            for(i=0; _lt(i,&PAD_BUTTON_COUNT;); i++){
                _clear_state(sButton[i]);
                sButton[i] = visible;
            }
        </script>
    </function>
    <function name="processInput" type="void">
        <comment>回答中のキー入力に対する処理</comment>
        <parameters>
            <var name="input" type="int"/>
        </parameters>
        <variables>
            <var name="i" type="int" />
            <var name="tmpNumber" type="int" />
        </variables>
        <script>
            // 入力された文字があっているか評価する
            tmpNumber = _string_to_int(
                mTrueAnswerChar[mIndex]);

            if(tmpNumber == input){
                // あたった！正解分文字列作成
                mBoxText = "";
                for(i=0; _lte(i,mIndex); i++){
                    mBoxText = _strcat(
                        mBoxText, mTrueAnswerChar[i]);
                }

                // 未正解部分は*を後ろにつなげる。
                for(i=0; _lt(i,&CHAR_SIZE; - mIndex -1); i++){
                    mBoxText = _strcat(mBoxText,"*");
                }

                // 途中経過を再描画
                _clear_state(sBox);
                sBox = true;

                if(mIndex == &CHAR_SIZE; - 1){//全部正解したら
                    // ボタンを入力できないようにする。
                    initButton(false);

                    mSeed = _msec();
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                    sStatus = 3;
                }
                mIndex++;
            }else{
                //間違ったら、画面背景を赤くする。
                mBackColor = mBadColor;
                _clear_state(sBackGround);
                sBackGround = true;
            }
        </script>

    </function>
</functions>

<script>
    mSeed = _msec();

    // スクリーンのサイズを調べる
    mScrWidth = _getIntProperty(&_PROPERTY_INT_SCREEN_WIDTH;);
    mScrHeight = _getIntProperty(&_PROPERTY_INT_SCREEN_HEIGHT;);

    // 開始ボタンの全文字数分の幅
    mStartWidth = (&SPACING; * 2) +
        _text_width("&START_MESSAGE;", &_FONT_SIZE_MEDIUM;,
        &_FONT_STYLE_BOLD;, &_FONT_FACE_SYSTEM;) +
        &BORDER_SIZE; * 2;

    // 桁全文字数分の幅 サイズを測るために initNumber を呼ぶ
    mAnswerWidth= (&SPACING; * 2) +
        _text_width(initNumber(), &_FONT_SIZE_MEDIUM;,
        &_FONT_STYLE_BOLD;, &_FONT_FACE_SYSTEM;) +
        &BORDER_SIZE; * 2;

    // 高さ：START ボタン及び解答欄
    mHeight = (&SPACING; * 2) +
        _text_height(&_FONT_SIZE_MEDIUM;, &_FONT_STYLE_ITALIC;,
        &_FONT_FACE_SYSTEM;) +
        &BORDER_SIZE; * 2;

    // ラベルをセンタリングする X は都度計算 Y のみここで初期化
    mPosY = (mScrHeight - mHeight) / 2;

    // 色の設定
    mBadColor = &_COLOR_RED;;
    mNormalColor = &_COLOR_WHITE;;
    mBackColor = mNormalColor;

    // 背景を描画
    sBackGround = true;

    // ゲーム開始
    sStatus = 0;
</script>
<display>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
        <fn>
            <text>終了</text>
            <event name="onselect">
                <accelerators><key>F1</key></accelerators>
                <script>
                    _unload();
                </script>
            </event>
        </fn>
    </display>
    <states>
        <state var="sBackGround">
            <transition value="true">
                <comment>背景の描画</comment>
                <display>
                    <box>
                        <width><eval>mScrWidth</eval></width>
                        <height><eval>mScrHeight</eval></height>
                        <fg><eval>mBackColor</eval></fg>
                        <bg><eval>mBackColor</eval></bg>
                    </box>
                </display>
                <delay>100</delay>
                <script>
                    if(mBackColor != mNormalColor){
                        mBackColor = mNormalColor;
                        sBackGround = false;
                        sBackGround = true;
                    }
                </script>
            </transition>
        </state>
        <state var="sStatus">
            <comment></comment>
            <transition value="0">
                <comment>START ボタン表示</comment>
                <script>
                    _clear_state(sBox);

                    mPosX = (mScrWidth - mStartWidth) / 2;
                    mBoxWidth = mStartWidth;
                    mBoxText = "&START_MESSAGE;";
                    sBox = true;
                </script>
            </transition>
            <transition value="1">
                <comment>数字を暗記する状態</comment>
                <script>
                    // 問題を作成し表示する
                    mBoxText = initNumber();
                    mPosX = (mScrWidth - mAnswerWidth) / 2;
                    mBoxWidth = mAnswerWidth;

                    // 問題の描画
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                _clear_state(sBox);
                sBox = true;

                // タイマーを実行する
                mTimer = &SEC_LIMIT;;
                _clear_state(sTimer);
                sTimer = true;
            </script>
        </transition>
        <transition value="2">
            <comment>解答する状態</comment>
            <variables>
                <var name="i" type="int"/>
            </variables>
            <script>
                // 最初に必要桁数だけ*を表示する。
                mBoxText = "";
                for(i=0;_lt(i,&CHAR_SIZE;);i++){
                    mBoxText = _strcat(mBoxText,"*");
                }
                // 解答欄の描画
                _clear_state(sBox);
                sBox = true;

                // 回答ボタンを使えるようにする
                initButton(true);

                mIndex = 0;
            </script>
        </transition>
        <transition value="3">
            <comment>おめでとう状態</comment>
            <script>
                // おめでとうメッセージを表示する
                _clear_state(sEnding);
                sEnding = true;
            </script>
        </transition>
    </state>
    <state var="sBox">
        <transition value="true">
            <comment>START ボタンと解答欄の実体</comment>
            <display>
                <!-- ボタンの枠 -->
                <box>
                    <x><eval>mPosX</eval></x>
                    <y><eval>mPosY</eval></y>
                    <width><eval>mBoxWidth</eval></width>
                    <height><eval>mHeight</eval></height>
                    <fg>&BORDER_COLOR;</fg>
                    <bg>&BORDER_COLOR;</bg>
                    <box>
                        <x><eval>&BORDER_SIZE;</eval></x>
                        <y><eval>&BORDER_SIZE;</eval></y>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                    <width>
                        <eval>mBoxWidth - &BORDER_SIZE; * 2</eval>
                    </width>
                    <height>
                        <eval>mHeight - &BORDER_SIZE; * 2</eval>
                    </height>
                    <fg>&BACK_COLOR;</fg>
                    <bg>&BACK_COLOR;</bg>
                    <event name="onSelect">
                        <accelerators><key>FIRE</key></accelerators>
                        <script>
                            // 出題中に遷移する
                            sStatus = 1;
                        </script>
                    </event>
                    <!-- 文字を表示する -->
                    <label>
                        <text><eval>mBoxText</eval></text>
                        <x>&SPACING;</x>
                        <y>&SPACING;</y>
                        <fg>&FONT_COLOR;</fg>
                        <style>&_FONT_STYLE_BOLD;</style>
                    </label>
                </box>
            </box>
        </display>
    </transition>
</state>
<state var="sTimer">
    <transition value="true">
        <display>
            <label>
                <text><eval>_strcat("残り :",mTimer,"秒")</eval></text>
                <fg>&_COLOR_BLACK;</fg>
                <bg>&_COLOR_WHITE;</bg>
            </label>
        </display>
        <delay>1000</delay>
        <script>
            sTimer = false;
            mTimer--;
            if(mTimer > 0){
                sTimer=true;
            }else{
                // 問題の解答を開始させる。
                sStatus = 2;
            }
        </script>
    </transition>
</state>

<state var="sButton" index="*">
    <comment>
    <!-- ボタンです：実際は透明なので画面上に表示されません。 -->
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <!-- イベント処理のためだけにあります。 -->
            </comment>
            <transition value="true">
                <comment>入力を受け付ける box 要素。画面表示無し</comment>
                <display>
                    <box>
                        <event name="onSelect">
                            <accelerators>
                                <key><eval>_state_index(0)</eval></key>
                            </accelerators>
                            <script>
                                processInput(_state_index(0));
                            </script>
                        </event>
                    </box>
                </display>
            </transition>
        </state>
        <state var="sEnding">
            <transition value="true">
                <comment>1 秒間のみ表示して終了</comment>
                <display>
                    <label>
                        <text>おめでとうございます。</text>
                        <fg>&_COLOR_BLACK;</fg>
                        <bg>&_COLOR_WHITE;</bg>
                    </label>
                </display>
                <delay>1000</delay>
                <script>
                    _clear_state(sEnding);
                    _clear_state(sStatus);
                    sStatus = 0;
                </script>
            </transition>
        </state>
    </states>
  </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

１０．画像の表示

本章では画像の表示について解説していきます。これまで紹介したヴィジュアル要素だけでなく、画像表示を組み合わせることでより視覚に訴えるアプリケーションを開発することができるでしょう。

本章ではこれまでに作成した記憶ゲームの一部を画像により装飾していきます。

簡単な画像の表示

UJML では<image>要素を使い画像を表示することができます。画像を表示するだけの基本的なコードを見てみましょう。

image.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <display>
            <image>
                <url><image-url>images/ome</image-url></url>
                <width>130</width>
                <height>30</height>
            </image>
        </display>
    </application>
</ujml>
```

<image>要素も基本的にはこれまで紹介したヴィジュアル要素と同じように記述します。<width>要素と<height>要素を正しく指定する必要があります。何も指定しない場合には画像は画面に表示されません。

<image>要素を使う際には気をつけるべきことがあります。デバイスによって描画できる画像ファイル形式（gif,jpeg,png 等）は異なります。あるデバイスでは gif 形式のみをサポートし、他のデバイスでは png 形式のみサポートされているかもしれません。アプリケーションを複数のデバイスに対応させたいときにはこれらのデバイスの差異を吸収する必要があります。そのために<image-url>要素を使います。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

この<image-url>要素は指定された URL に対して、デバイスが対応しているファイル形式の拡張子を自動的に補足してくれます。ここでは、

```
<url><image-url>images/ome</image-url></url>
```

のように記述し、ファイル名に拡張子をつけていません。このように記述することで、gif 形式をサポートしたデバイスの場合には images/ome.gif を、png 形式をサポートしたデバイスの場合には images/ome.png を表示します。

アプリケーションを開発する際には、様々な種類のデバイスを意識して画像ファイルも画像ファイル形式の分だけ用意する必要があります。今回は、images フォルダの中に、

ome.bmp
ome.gif
ome.jpg
ome.png

の四種類のファイルを作成しました。

ただし、<image-url>は必ずしも使う必要はありません。もし、gif 形式をサポートした限定的なデバイスのみへアプリケーションを提供するなら、<image-url>要素を使わずに

```
<url>images/ome.gif</url>
```

と記述することもできます。

リソースの利用

先ほど紹介した image.ujml では画像が画面に表示されました。ところで、画像はデバイスにいつ取り込まれるのでしょうか？UJML ソースコードをコンパイルしてできるバイトコード中に画像ファイルは含まれません。image.ujml では画像ファイルは<image>要素の表示処理が実行されたときに、アプリケーションの実行環境に取り込まれます。

いくつかのケースではこのように表示の実行と同時に画像ファイルが取り込まれるのは不都合な場合があります。例えば、画像ファイルがネットワーク上にありネットワークの遅延等で期待通りにファイルがすぐに取り込まれないかもしれません。UJML ではそのような場合に備え、画像ファイルを事前にロードする仕組が提供されています。それが

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

<resource>要素です。では、<resource>要素の使い方を見てみましょう。

resource.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sImage" type="boolean"/>
            <state-var name="sLoad" type="boolean"/>
        </state-variables>
        <script>
            // リソースの読み込みを開始する。
            sLoad = true;
        </script>
        <states>
            <state var="sLoad">
                <transition value="true">
                    <resources>
                        <resource>
                            <url><image-url>images/ome</image-url></url>
                            <event name="onResourceAvailable">
                                <script>
                                    // 画像の読み込みに成功、画像を表示
                                    sImage = true;
                                </script>
                            </event>
                        </resource>
                    </resources>
                </transition>
            </state>
            <state var="sImage">
                <transition value="true">
                    <display>
                        <image>
                            <url><image-url>images/ome</image-url></url>
                            <width>130</width>
                            <height>30</height>
                        </image>
                    </display>
                </transition>
            </state>
        </states>
    </application>
</ujml>
```

resource.ujmlではステート変数sLoadをアプリケーション実行時のデフォルトスクリプト内で true に設定し、リソースのロードを開始させています。リソースのロードに成功すると、onResourceAvailable イベントが発生します。この時点で画像ファイルのロードが成功

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

していますので、ステート変数 sImage を true に設定し画像を画面に表示させています。

onResourceAvailable イベントは該当するリソースが使用できることを保証します。

また、<resource>要素の親要素には必ず<transition>要素があります（親の親として）。つまり、リソースをロードするためには必ずその状態を管理するためのステート変数が必要です。

では、このステート変数とリソースの関係を簡単に解説します。resource.ujml でリソースのロード開始に使用するステート変数 sLoad は実はロードの開始だけを担当するわけではありません。実は見えないところで他の役割も果たしています。

onResourceAvailable イベントが発生したとき、リソースはすぐに使える状態になっています。ここで、すぐに使える状態とはデバイスのメモリにリソースが展開されている状態を指します。この状態でステート変数 sImage を true に設定すると最も早く画面上に画像を表示することができます。

ここで、sLoad を false にすると、メモリ中に展開されていたリソースがメモリからの削除対象候補となります。

ここで敢えて「候補」と書いたのは実際にメモリからただちに削除されるとは限らず、メモリからの削除は UIEPlayer が管理しているからです。例えば、しばらく使わない画像はメモリから削除することでアプリケーションの動作の効率化を図ることが可能です。

ワンポイント：ステート変数によりリソースを適宜管理しメモリの有効利用を心がけましょう。

ところで、何らかの理由でリソースが正しくロードできるとは限りません。もし、リソースのロードに失敗した場合には、onResourceError というイベントが発生します。このイベントに対する処理も必ず記述することを推奨します。このイベントの処理でたとえば、「予期せぬエラーにより画像ファイルをロードできませんでした。」といったエラーメッセージを表示させることも可能です。

resource.ujml を若干修正して、リソースのロードに失敗するケースも考慮した簡単なサンプルを見てみましょう。この resource_error.ujml を実行する際にはアプリケーションが参照する画像ファイルを削除してリソースのロードが必ず失敗するようにします。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

resource_error.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd">
<ujml>
    <application>
        <state-variables>
            <state-var name="sImage" type="int"/>
            <state-var name="sLoad" type="boolean"/>
        </state-variables>
        <script>
            // リソースの読み込みを開始する。
            sLoad = true;
        </script>
        <states>
            <state var="sLoad">
                <transition value="true">
                    <resources>
                        <resource>
                            <url><image-url>images/ome</image-url></url>
                            <event name="onResourceAvailable">
                                <script>
                                    // 画像の読み込みに成功、画像を表示
                                    sImage = 1;
                                </script>
                            </event>
                            <event name="onResourceError">
                                <script>
                                    // 画像の読み込みに失敗
                                    sImage = 2;
                                </script>
                            </event>
                        </resource>
                    </resources>
                </transition>
            </state>
            <state var="sImage">
                <transition value="1">
                    <display>
                        <image>
                            <url><image-url>images/ome</image-url></url>
                            <width>130</width>
                            <height>30</height>
                        </image>
                    </display>
                </transition>
                <transition value="2">
                    <display>
                        <label>
                            <text>おめでとうございます！</text>
                        </label>
                    </display>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            </transition>
        </state>
    </states>
</application>
</ujml>
```

ここでは、ステート変数 sImage を int 型で宣言しています。リソースの読み込みに成功したらステート変数 sImage の値を 1 に設定し、失敗した場合にはステート変数 sImage の値を 2 に設定しています。ステート変数 sImage が 2 の場合の<transition>要素内には<image>要素の代わりに<label>要素を定義し、文字として「おめでとうございます！」を表示するようにしています。

このようにすることで、万が一正しく画像ファイルが読み込めなかった場合でもアプリケーションの実行を継続させることもできるでしょう。仮に、リソースが読み込めないことがアプリケーションにとって致命的な場合には、「予期せぬエラーによりアプリケーションを続行できません。しばらくたってから再度実行するか、このエラーが何度も繰り返す場合には○○までお問い合わせください。」といったメッセージを表示することも可能です。ユーザーが迷わないような実装を推奨します。

画像によるおめでとうメッセージの表示

それでは、完成した記憶ゲームの一部を画像化してみましょう。画像化するのは、正解したときのおめでとうメッセージです。<resource>要素で画像が正しく読み込めなかった場合を考慮して、通常の文字表示も残す形で実装します。

main_image.ujml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ujml PUBLIC "-//UIEVOLUTION//DTD UJML 2.1//EN"
"http://www.uievolution.com/dtd/ujml-2.1.dtd" [

    <!-- 表示文字関係の定数 -->
    <!ENTITY START_MESSAGE "START" >

    <!-- デザイン関係の定数 -->
    <!ENTITY SPACING "4">
    <!ENTITY BORDER_SIZE "2">
    <!ENTITY BORDER_COLOR "&_COLOR_BLUE;">
    <!ENTITY BACK_COLOR "&_COLOR_SILVER;">
    <!ENTITY FONT_COLOR "&_COLOR_BLACK;">

    <!-- 動作関係の定数 -->
    <!ENTITY SEC_LIMIT "10"><!-- 覚える秒数 -->
    <!ENTITY PAD_BUTTON_COUNT "10">
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
    <!ENTITY CHAR_SIZE "10" ><!-- 問題の桁数 -->

]>
<ujml>
    <application>
        <state-variables>
            <state-var name="sBackGround" type="boolean"/>
            <state-var name="sStatus" type="int"/>

            <state-var name="sLoad" type="boolean"/>
            <state-var name="sTimer" type="boolean"/>
            <state-var name="sBox" type="boolean"/>
            <state-var name="sButton" type="boolean" size="&PAD_BUTTON_COUNT;"/>
            <state-var name="sEnding" type="int"/>

        </state-variables>
        <variables>
            <var name="mScrWidth" type="int" />
            <var name="mScrHeight" type="int" />

            <var name="mPosX" type="int" />
            <var name="mPosY" type="int" />

            <var name="mStartWidth" type="int"/>
            <var name="mHeight" type="int"/>

            <var name="mTimer" type="int"/>
            <var name="mSeed" type="int"/>
            <var name="mIndex" type="int"/><!-- 解答の位置 -->

            <var name="mBoxWidth" type="int"/><!-- 解答欄等の幅 -->
            <var name="mBoxText" type="string"/><!-- 解答欄に表示する文字 -->

            <var name="mAnswerWidth" type="int"/>
            <var name="mPosXAnswer" type="int"/>
            <!-- 正解の文字 -->
            <var name="mTrueAnswerChar" type="string" size="&CHAR_SIZE;"/>

            <var name="mBackColor" type="int"/>
            <var name="mBadColor" type="int"/>
            <var name="mNormalColor" type="int"/>

            <var name="mImageReady" type="boolean"/>
        </variables>
        <functions>
            <function name="randRange" type="int">
                <comment>指定範囲の数字をランダムに作る関数</comment>
                <parameters>
                    <var name="seed" type="int"/>
                    <var name="min" type="int"/>
                    <var name="max" type="int"/>
                </parameters>
                <return>
                    <eval>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                (_srand(seed) % (max - min + 1)) + min
            </eval>
        </return>
    </function>
    <function name="initNumber" type="string">
        <comment>問題の数値を作成する関数</comment>
        <variables>
            <var name="tmpSeed" type="int"/>
            <var name="tmpNumber" type="string"/>
            <var name="i" type="int"/>
        </variables>
        <script>
            tmpSeed = _msec();
            tmpNumber = "";
            for(i=0; _lt(i,&CHAR_SIZE;); i++){
                tmpNumber = _strcat(tmpNumber, randRange(tmpSeed,0,9));
                mTrueAnswerChar[i] = randRange(tmpSeed,0,9);
                tmpSeed = _srand(tmpSeed);
            }
        </script>
        <return>
            <eval>tmpNumber</eval>
        </return>
    </function>
    <function name="initButton" type="void">
        <comment>0～9 の入力ボタンを初期化する</comment>
        <parameters>
            <var name="visible" type="boolean"/>
        </parameters>
        <variables>
            <var name="i" type="int"/>
        </variables>
        <script>
            for(i=0; _lt(i,&PAD_BUTTON_COUNT;); i++){
                _clear_state(sButton[i]);
                sButton[i] = visible;
            }
        </script>
    </function>
    <function name="processInput" type="void">
        <comment>回答中のキー入力に対する処理</comment>
        <parameters>
            <var name="input" type="int"/>
        </parameters>
        <variables>
            <var name="i" type="int" />
            <var name="tmpNumber" type="int" />
        </variables>
        <script>
            // 入力された文字があっているか評価する
            tmpNumber = _string_to_int(
                mTrueAnswerChar[mIndex]);

            if(tmpNumber == input){
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                // あたった！正解分文字列作成
                mBoxText = "";
                for(i=0; _lte(i,mIndex); i++){
                    mBoxText = _strcat(
                        mBoxText, mTrueAnswerChar[i]);
                }

                // 未正解部分は*を後ろにつなげる。
                for(i=0; _lt(i,&CHAR_SIZE; - mIndex -1); i++){
                    mBoxText = _strcat(mBoxText,"*");
                }

                // 途中経過を再描画
                _clear_state(sBox);
                sBox = true;

                if(mIndex == &CHAR_SIZE; - 1){//全部正解したら
                    // ボタンを入力できないようにする。
                    initButton(false);

                    mSeed = _msec();
                    sStatus = 3;
                }
                mIndex++;
            }else{
                //間違ったら、画面背景を赤くする。
                mBackColor = mBadColor;
                _clear_state(sBackGround);
                sBackGround = true;
            }
        </script>

    </function>
</functions>

<script>
    mSeed = _msec();

    // スクリーンのサイズを調べる
    mScrWidth = _getIntProperty(&_PROPERTY_INT_SCREEN_WIDTH;);
    mScrHeight = _getIntProperty(&_PROPERTY_INT_SCREEN_HEIGHT;);

    // 開始ボタンの全文字数分の幅
    mStartWidth = (&SPACING; * 2) +
        _text_width("&START_MESSAGE;", &_FONT_SIZE_MEDIUM;,
        &_FONT_STYLE_BOLD;, &_FONT_FACE_SYSTEM;) +
        &BORDER_SIZE; * 2;

    // 桁全文字数分の幅 サイズを測るために initNumber を呼ぶ
    mAnswerWidth= (&SPACING; * 2) +
        _text_width(initNumber(), &_FONT_SIZE_MEDIUM;,
        &_FONT_STYLE_BOLD;, &_FONT_FACE_SYSTEM;) +
        &BORDER_SIZE; * 2;
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
        // 高さ：START ボタン及び解答欄
        mHeight = (&SPACING; * 2) +
            _text_height(&_FONT_SIZE_MEDIUM;, &_FONT_STYLE_ITALIC;,
            &_FONT_FACE_SYSTEM;) +
            &BORDER_SIZE; * 2;

        // ラベルをセンタリングする X は都度計算 Y のみここで初期化
        mPosY = (mScrHeight - mHeight) / 2;

        // 色の設定
        mBadColor = &_COLOR_RED;;
        mNormalColor = &_COLOR_WHITE;;
        mBackColor = mNormalColor;

        // 背景を描画
        sBackGround = true;

        // 画像の読み込み開始
        sLoad = true;

    </script>
    <display>
        <fn>
            <text>終了</text>
            <event name="onselect">
                <accelerators><key>F1</key></accelerators>
                <script>
                    _unload();
                </script>
            </event>
        </fn>
    </display>
    <states>
        <state var="sLoad">
            <transition value="true">
                <resources>
                    <resource>
                        <url><image-url>images/ome</image-url></url>
                        <event name="onResourceAvailable">
                            <script>
                                // 画像の読み込みに成功
                                mImageReady = true;
                                // ゲーム開始
                                sStatus = 0;
                            </script>
                        </event>
                        <event name="onResourceError">
                            <script>
                                // 画像の読み込みに失敗
                                mImageReady = false;
                                // ゲーム開始
                                sStatus = 0;
                            </script>
                        </event>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                </resource>
            </resources>
        </transition>
    </state>
    <state var="sBackGround">
        <transition value="true">
            <comment>背景の描画</comment>
            <display>
                <box>
                    <width><eval>mScrWidth</eval></width>
                    <height><eval>mScrHeight</eval></height>
                    <fg><eval>mBackColor</eval></fg>
                    <bg><eval>mBackColor</eval></bg>
                </box>
            </display>
            <delay>100</delay>
            <script>
                if(mBackColor != mNormalColor){
                    mBackColor = mNormalColor;
                    sBackGround = false;
                    sBackGround = true;
                }
            </script>
        </transition>
    </state>
    <state var="sStatus">
        <comment></comment>
        <transition value="0">
            <comment>START ボタン表示</comment>
            <script>
                _clear_state(sBox);

                mPosX = (mScrWidth - mStartWidth) / 2;
                mBoxWidth = mStartWidth;
                mBoxText = "&START_MESSAGE;";
                sBox = true;
            </script>
        </transition>
        <transition value="1">
            <comment>数字を暗記する状態</comment>
            <script>
                // 問題を作成し表示する
                mBoxText = initNumber();
                mPosX = (mScrWidth - mAnswerWidth) / 2;
                mBoxWidth = mAnswerWidth;

                // 問題の描画
                _clear_state(sBox);
                sBox = true;

                // タイマーを実行する
                mTimer = &SEC_LIMIT;;
                _clear_state(sTimer);
                sTimer = true;
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            </script>
        </transition>
        <transition value="2">
            <comment>解答する状態</comment>
            <variables>
                <var name="i" type="int"/>
            </variables>
            <script>
                // 最初に必要桁数だけ*を表示する。
                mBoxText = "";
                for(i=0;_lt(i,&CHAR_SIZE;);i++){
                    mBoxText = _strcat(mBoxText,"*");
                }
                // 解答欄の描画
                _clear_state(sBox);
                sBox = true;

                // 回答ボタンを使えるようにする
                initButton(true);

                mIndex = 0;
            </script>
        </transition>
        <transition value="3">
            <comment>おめでとう状態</comment>
            <script>
                // おめでとうメッセージを表示する
                _clear_state(sEnding);
                // 画像が表示できるかにより分岐
                if(mImageReady){
                    sEnding = 1;
                }else{
                    sEnding = 2;
                }
            </script>
        </transition>
    </state>
    <state var="sBox">
        <transition value="true">
            <comment>START ボタンと解答欄の実体</comment>
            <display>
                <!-- ボタンの枠 -->
                <box>
                    <x><eval>mPosX</eval></x>
                    <y><eval>mPosY</eval></y>
                    <width><eval>mBoxWidth</eval></width>
                    <height><eval>mHeight</eval></height>
                    <fg>&BORDER_COLOR;</fg>
                    <bg>&BORDER_COLOR;</bg>
                    <box>
                        <x><eval>&BORDER_SIZE;</eval></x>
                        <y><eval>&BORDER_SIZE;</eval></y>
                        <width>
                            <eval>mBoxWidth - &BORDER_SIZE; * 2</eval>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
                    </width>
                    <height>
                        <eval>mHeight - &BORDER_SIZE; * 2</eval>
                    </height>
                    <fg>&BACK_COLOR;</fg>
                    <bg>&BACK_COLOR;</bg>
                    <event name="onSelect">
                        <accelerators><key>FIRE</key></accelerators>
                        <script>
                            // 出題中に遷移する
                            sStatus = 1;
                        </script>
                    </event>
                    <!-- 文字を表示する -->
                    <label>
                        <text><eval>mBoxText</eval></text>
                        <x>&SPACING;</x>
                        <y>&SPACING;</y>
                        <fg>&FONT_COLOR;</fg>
                        <style>&_FONT_STYLE_BOLD;</style>
                    </label>
                </box>
            </box>
        </display>
    </transition>
</state>
<state var="sTimer">
    <transition value="true">
        <display>
            <label>
                <text><eval>_strcat("残り :",mTimer,"秒")</eval></text>
                <fg>&_COLOR_BLACK;</fg>
                <bg>&_COLOR_WHITE;</bg>
            </label>
        </display>
        <delay>1000</delay>
        <script>
            sTimer = false;
            mTimer--;
            if(mTimer > 0){
                sTimer=true;
            }else{
                // 問題の解答を開始させる。
                sStatus = 2;
            }
        </script>
    </transition>
</state>

<state var="sButton" index="*">
    <comment>
    <!-- ボタンです：実際は透明なので画面上に表示されません。 -->
    <!-- イベント処理のためだけにあります。 -->
    </comment>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

```
            <transition value="true">
                <comment>入力を受け付ける box 要素。画面表示無し</comment>
                <display>
                    <box>
                        <event name="onSelect">
                            <accelerators>
                                <key><eval>_state_index(0)</eval></key>
                            </accelerators>
                            <script>
                                processInput(_state_index(0));
                            </script>
                        </event>
                    </box>
                </display>
            </transition>
        </state>
        <state var="sEnding">
            <transition value="1">
                <comment>1 秒間のみ表示して終了</comment>
                <display>
                    <image>
                        <url><image-url>images/ome</image-url></url>
                        <width>130</width>
                        <height>30</height>
                    </image>
                </display>
                <delay>1000</delay>
                <script>
                    _clear_state(sEnding);
                    _clear_state(sStatus);
                    sStatus = 0;
                </script>
            </transition>
            <transition value="2">
                <comment>1 秒間のみ表示して終了</comment>
                <display>
                    <label>
                        <text>おめでとうございます。</text>
                        <fg>&_COLOR_BLACK;</fg>
                        <bg>&_COLOR_WHITE;</bg>
                    </label>
                </display>
                <delay>1000</delay>
                <script>
                    _clear_state(sEnding);
                    _clear_state(sStatus);
                    sStatus = 0;
                </script>
            </transition>
        </state>
    </states>
  </application>
</ujml>
```

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

まず、デフォルトスクリプトでステート変数 sLoad を true に設定します。これで、<resource>要素による画像ファイルの読み込みが開始されます。さらに、画像ファイルの読み込み成功により onResourceAvailable イベントが発生し、ステート変数 sStatus が 0 に設定されゲームが開始されます。この際、画像が正しく読み込めたことを示す変数 mImageReady を true にしています。

もしここで、仮に画像ファイルの読み込みに失敗すると、onResourceError イベントが発生し、mImageReady が false となったまま、ステート変数 sStatus が 0 に設定されゲームが開始されます。今回は「画像が表示されなくても文字を表示すれば良い」という方針で画像ファイルの読み込みに失敗してもゲームを開始しています。

その後、問題に正解した時にステート変数 sEnding に値が設定されて「おめでとう」メッセージが表示されます。これまでは、sEnding は boolean 型のステート変数でしたが、今回は int 型に変更しています。これは、画像ファイルが読み込めたかどうかを示す変数 mImageReady により、画像を表示するか、文字を表示するかという分岐を行うためです。

おめでとうメッセージを表示する際次の表の条件にしたがってステート変数 sEnding に値を設定します。

状態	mImageReady	sEnding に設定する値
画像が読み込みに成功	true	1
画像の読み込みに失敗	false	2

ステート変数 sEnding に 1 が設定された場合には<image>要素が表示されるようにし、ステート変数 sEnding に 2 が設定された場合には<label>要素が表示されます。

本来であれば、<resource>要素により画像の読み込みが開始され終了するまでの時間を考え、読み込み中に「Loading」等のメッセージを表示した方が良いでしょう。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

付録

開発者向けティップス、注意事項

実際に UJML での開発に携わっているエンジニアの方に話を伺いいくつかの開発者向けのティップスを収録しました。内容には本ドキュメントの範囲を超える内容も含まれていますが、一度目を通すことをおすすめします。

XML 要素の順番に気をつけよう

UJML は子要素の順序を厳密に規定しています。例えば<box>要素の子要素として<x>要素、<y>要素がありますが、順序は<x>要素、次に<y>要素の順であり、逆の順序にするとコンパイル時にエラーが発生します。良く使う要素の順序は覚えるか、最初のうちはリファレンスを手元おいて作業するようにしましょう。

文字列に改行コードを含めたい

<multi-label>要素で複数行の文字列を表示することができます。途中で改行したいときには、次のようにして改行コードを挿入することができます。

まず、ENTITY として

```
<!ENTITY NEWLINE "&#000A;">
```

を定義します。次に改行を含めたいところに&NEWLINE;を挿入します。

ソースコードは自分で最適化しよう

コンパイラにはプリプロセッサの機能がありません。コンパイラはソースコードを自動的に最適化しませんので開発者が最適なソースコードを記述する必要があります。例えば、

```
a = 1 + 1
```

はコンパイル時 2 とならず、実行時に計算されます。ソースコード中に明示的に

```
a=2
```

と書きましょう。良く、時間の計算をするために、ソースコード中に

```
ds = 60 * 60 :* 24;
```

といった具合に記述することがありますが、必ず計算結果 86400 をソースコード中に記述しましょう。

画像ファイルはひとつにまとめて節約しよう

UJML では画像ファイルの部分表示が可能です。複数の画像が必要な場合には、できるだけ一つのファイルにまとめた方がリソースを節約することができます。例えば、キャラクターの向きのイメージを別々の画像ファイルとして用意するのではなく、一つの画像ファイルとしてまとめた方がリソースを節約できます。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

string 型と int 型のキャストについて

スクリプト中では、int 型から string 型へのキャストは自動的に行われますが、string 型から int 型へキャストするときには、関数_string_to_int()を使う必要があります。

画像ファイルはあらかじめ<resource>要素を使いロードしよう

画像ファイルを表示させる場合にはできるだけ<resource>要素を使い事前にロードできるように管理した方が良いです。<resource>要素による事前ロードをせずに、<image>要素だけで画像表示をすると期待通りに画像が表示されない可能性があります。

<resource>要素を使うときはエラーハンドリングをしっかりしよう

<resource>要素で画像等をロードする時、ネットワーク上の障害で正しくロードできない可能性があります。必ずエラーハンドリングをし、アプリケーションがリソースの状態を管理できるように実装しましょう。

アプリケーションのメインファイルは main.ujml という名前にしよう

アプリケーションのメインファイルをわかりやすくするために、main.ujml を使うことを推奨します。

透明<box>を使いイベントの処理をしよう

描画されている要素に依存しないような処理は透明<box>要素を使ってイベント処理するようにしましょう。その他のヴィジュアル要素を使っても同様のことは実現可能ですが、<box>要素を使い統一することを推奨します。

描画更新が必要なオブジェクト毎にステート変数を宣言しよう

再描画が必要なヴィジュアル要素には（例えば<label>要素の文字を度々変化させる等）、ステート変数を個別に宣言しましょう。複数のヴィジュアル要素を一つのステート変数で再描画することもできますが、その中に再描画する必要の無いものが含まれるとリソースを無駄に消費してしまいます。

ヴィジュアル要素は親子間で座標が相対座標となることに注意しよう

ヴィジュアル要素（<box>要素、<label>要素等）は親要素との間で座標が相対座標になります。親要素の左上の頂点が原点座標です。これを忘れていると、ときどき表示位置が画面外となり何も表示されないことにとまどうので気をつけましょう。

繰り返し文 while,for で break/continue が使える。

UJML2.1 から繰り返し制御文の while,for で break/continue 命令を使うことができるようになっています。

ビット演算子がありません

ビット演算子は標準で提供されていません。ビット演算をする場合には自分で実装する必要があります。

チームで開発するときには、ソースコードの記述ルールを決めよう

UJML ではステート変数を使いアプリケーションの状態を管理でき大変便利です。ただ、<transition>要素で定義するアプリケーションの状態や実行内容がソースコード中に偏在することになります。そのため慣れるまで他の人が書いたソースコードを読むのに時間がかかります。チームで開発するときには、ソースコードの記述ルールを決めそれに沿った開発を進めましょう。

Copyright (c) 2007 UIE Japan, Inc.　無断転載禁止

開発環境（デバッガ等）の注意事項

ここからは開発環境 UIEDeveloper に関する内容です。

開発生産性の向上させるために XML エディタを使おう
UIEDevelopper に付属しているエディタよりも強力なエディタがエクリプスプラグインとして提供されているかもしれません。開発生産性を高めるために、タグ保管、自動ヴァリデーション機能のついた XML エディタを使うことをお勧めします。エクリプスプラグインとして例えば xmlbuddy という XML エディタがあります。 http://xmlbuddy.com からダウンロードすることができます。動作については保証できませんので自己責任でお願いいたします。

SDK は Windows および Mac OS のみ
SDK は現状では Windows および Mac OS 環境のみ動作保証しています。Linux 等での動作は保証されていません。ご注意ください。

SDK はオンラインでアップデートしよう
SDK は SDK から直接アップデートすることができます。オンラインアップデートにより、UIEPlayer パックの最新版の入手や、サンプルの最新版の入手が可能です。アップデートは SDK のメニュー「Help」→「Software Updates」→「Find and Install」から行えます。

より高度なアプリケーション設計に向けて

パーティションについて

複数の HTML ページからウェブサイトを構築できるのと同様に、UJML アプリケーションを複数のパーティションから構成することが可能です。

他のパーティションを動的に読み込むには 2 つの方法があります。_run()関数の使用と_link()関数の使用の 2 つの方法です。_run() 関数によって読み込むパーティションはアプリケーションの「メインパーティション」あるいは「親パーティション」と呼ばれます。_link() 関数で読み込み、_unlink() 関数でアンロードするパーティションは子パーティションと呼ばれます。

_run()関数

下の図は、App 1 をデフォルトのアプリケーションを起動し、App 1 が App 2 を実行したときのヒストリースタックを模式的に表しています。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

ヒストリースタック
3:
2:
1:App 2
0:App 1

App 1 はアプリケーション内で、

```
<script>
   _run(App 2);
</script>
```

のようにしてパーティション App 2 を実行します。この時点で重要なのは、App 1 がメモリを物理的に占有していないということです。また、この時点でほとんどの UIEPlayer は App 1 をキャッシュしています。そうすることで、App 2 がアンロードされたときに App 1 をネットワーク経由で取得しないように済むようにしています。

その後、App 2 が_unload()関数を呼び、自分自身をアンロードすると、App 1 が再起動されヒストリースタックは次の図のようになります。

ヒストリースタック
3:
2:
1:
0:App 1

_link()関数

_link() 関数でリンクしたパーティションはヒストリースタックに追加されません。その代わり、このパーティションは呼び出し元の子パーティションとなり、親（呼び出し元）がアンロードされると共にアンロードされます。

先ほどのヒストリースタックの例に倣うと、App 2 を実行された時、ヒストリースタックは下図のようになっています。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

ヒストリースタック
3:
2:
1:App 2
0:App 1

ここで、App 2 が別のパーティション Part A をリンクします。その場合でもヒストリースタックには Part A は格納されません。App 2 及び Part A は共に実行されます。

メモリリソースを使用するパーティション

メモリリソースを使用するパーティションはヒストリースタックの一番上にある現在のアプリケーションと、そこからリンクされているパーティションに限られます。アプリケーションがヒストリースタックの一番上でなくなった時に、そのアプリケーションが使用するすべてのメモリリソースは解放され、新しいアプリケーションによる使用が可能になります。

_run() 関数を呼びアプリケーションがヒストリースタックの一番上にロードされている間は、ロードが完了するまで現在のアプリケーションがメモリリソースを消費します。アプリケーションを開発する際は、新旧アプリケーションの両方がメモリに同時に収まるように設計してください。

この問題を回避する方法の一つに、メモリの消費量が少ない中間アプリケーション（ローダー）を使用する方法があります。

ディスプレイテンプレートの活用

UJML では「ディスプレイテンプレート」というテンプレートを定義することができます。ディスプレイテンプレートは次の二つの目的で使用される場合が多いです。

- アプリケーションサイズを小さくする場合
- 再利用可能なディスプレイ要素を定義する場合

UJML アプリケーション開発においてデバイスによる制限からアプリケーションサイズを抑えたいことは良くあることです。その際、見た目上全く同じものを複数個画面に表示する場合にはディスプレイテンプレートを使用した方がアプリケーションサイズを小さく

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止

できる可能性があります。例えば、グリッド表示させるための多数の<box>要素があてはまります。

オブジェクト指向によるアプリケーション設計

UJML ではオブジェクト指向に基づくアプリケーション設計が可能です。一般にオブジェクト指向と呼ばれるには次の 3 つの要素が含まれています。

- カプセル化
- ポリモーフィズム
- 継承

しかしながら、UJML が現バージョン 2.1 でサポートしているのはカプセル化とポリモーフィズムだけです。UJML のオブジェクト指向については、<interface>要素、<component>要素といった具体的な解説を見ていただくのが良いでしょう。ここでは、紹介にとどめます。

Copyright (c) 2007 UIE Japan, Inc. 無断転載禁止